Autorizada la entrega del proyecto del alumno:

Antonio Francisco Urbano Muñoz

EL DIRECTOR DEL PROYECTO

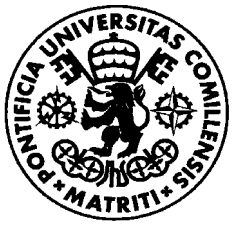
José María Rodríguez Fernández

Fdo.: Fecha:……/ ……/……

Vº Bº del Coordinador de Proyectos

Miguel Ángel Sanz Bobi

Fdo.: Fecha:……/ ……/……



UNIVERSIDAD PONTIFICIA COMILLAS

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

INGENIERO EN INFORMÁTICA

PROYECTO FIN DE CARRERA

DESARROLLO DE UN PROGRAMA INTELIGENTE DE APRENDIZAJE Y EVALUACIÓN AUTÓNOMA Y COMPETITIVA DEL JUEGO DE ESTRATEGIA GO

AUTOR: **Antonio Francisco Urbano Muñoz**

MADRID, Junio de 2007

DESARROLLO DE UN PROGRAMA INTELIGENTE DE APRENDIZAJE Y EVALUEACIÓN AUTÓNOMA Y COMPETITIVA DEL JUEGO DE ESTRATEGIA GO

Autor: Urbano Muñoz, Antonio Francisco.

Director: Rodríguez Fernández, José María.

Entidad Colaboradora: ICAI.- Universidad Pontificia de Comillas.

RESUMEN DEL PROYECTO

El proyecto consiste en la realización de una aplicación inteligente que, partiendo de las seis reglas básicas del Go y un limitado conjunto de estrategias "aprendidas a priori" (reglas fijas o restricciones), permita a un jugador novel realizar un aprendizaje progresivo, con evaluación autónoma y competitiva.

Este proyecto causará un impacto en los juegos de estrategia, introduciendo en este mercado el primer programa inteligente de aprendizaje para el juego del Go.

Para realizar este proyecto se ha desarrollado un interfaz agradable y cómoda del juego para el usuario, algoritmos para la aplicación de las reglas del juego, un sistema experto que implemente las estrategias básicas del juego, un interfaz gráfico para la creación de problemas, se ha cumplido la planificación especificada, se ha añadido la gestión de problemas y de de perfiles de usuarios de formar gráfica.

Se utilizará el modelo de desarrollo evolutivo. En el modelo evolutivo, los requerimientos son cuidadosamente examinados, y sólo esos que son bien comprendidos son seleccionados para el primer incremento. Los desarrolladores construyen una implementación parcial del sistema que recibe sólo estos requerimientos. Mientras se desarrolla el sistema, los usuarios lo usan, y proveen retroalimentación a los desarrolladores. Basada en esta retroalimentación, la especificación de requerimientos es actualizada, y una segunda versión del producto es desarrollada y desplegada. El proceso se repite indefinidamente.

El modelo que se va a seguir para la gestión de la base de datos, será el modelo relacional. La información puede ser recuperada o almacenada por medio de «consultas» que ofrecen una amplia flexibilidad y poder para administrar la información.

Se ha utilizado como lenguaje de programación Java y como entorno de desarrollo Eclipse, para almacenar toda la información referente a los problemas y a los perfiles de usuario MySql, para el desarrollo de la documentación se ha utilizado Microsoft Word 2007.

DEVELOP OF AN INTELLIGENT GO'S STRATEGY GAME PROGRAM THAT HELPS LEARNING, AUTO EVALUATES AND COMPETES

ABSTRACT

This project consists in developing an application that uses artificial intelligence. The application uses the six basic rules of the Go game and a set of strategies, this allows novice players to learn progressively, with independent and competitive evaluation.

This project will have high impact in strategy games, introducing into the market the first intelligence application for learning the Go game.

The project development has had various parts; those parts are a friendly user interface for the game, an algorithm to apply the game rules, an expert system to implement the basic strategies of the game and a friendly user interface for editing problems.

The evolutionary development is the development methodology used. The method of evolutionary development examines requirements carefully before proceeding to the next stage. The requirements that are understood are selected for the first version of the application. While development is carried out, the users test the different prototypes and provide backup information and new requirements to developers. Developers based on those requirements build newer and more complete prototypes of the product. This process is repeated until the client accepts the prototype as the one that fulfills all the requirements.

The database model has been developed using the entity relationship scheme. The information can be recovered and stored with different queries. Those queries provide flexibility to manage information.

The Java programming language is the one used in the project and the development environment used has been eclipse.

To store all the information of the problems and the user profiles MySQL database has been used.

The documentation and management of the project has been done with Microsoft Office 2007.

Contenido

1. Introducción 1
1.1 Breve introducción histórica del juego del Go 1
1.2 Características básicas del Go 2
1.3 Las seis reglas básicas del Go 2
1.4 Estrategias básicas del Go 5
2. Objetivos del proyecto 6
2.1 Estructura de la división del trabajo (EDT). 7
2.2 Organización 11
3. Diseño de la aplicación 13
3.1 Metodologías utilizadas 13
3.2 Identificación de necesidades 15
3.2.1 Objetivos de la aplicación 15
3.2.2 Alcance de la aplicación 16
3.3 Análisis de requisitos 16
3.3.1 Reconocimiento del problema 16
3.3.2 Creación de la lista de requisitos 17
3.3.3 Casos de uso 19
3.3.4 Modelo conceptual de datos 30
3.4 Diseño del juego Go 37
3.5 Diseño del editor de problemas 45
3.6 Diseño de la gestión 52
3.7 Diseño de la base de datos 62
4. Programación de prototipos 67
4.1. Prototipo I 68
4.2. Prototipo II 70
4.3. Prototipo III 75
4.4. Prototipo IV 85
4.5 Prototipo V 89
5. Herramientas utilizadas 93
6. Pruebas 94
7. Planificación 95
8. Presupuesto 96
9. Conclusiones 97
10. Bibliografía 98
Anexo A - Manual de Instalación I
Anexo B - Manual de Usuario I

1. Introducción [WIKI07], [ANRI04]

1.1 Breve introducción histórica del juego del Go

El Go es probablemente el juego de tablero más antiguo del mundo, nacido en China hace más de 4.000 años. Se dice que fue el primer Emperador de China, figura mitológica, quien inventó el juego para mejorar el desarrollo mental de un hijo de "pocas luces".

Como el Ajedrez, el Go es un juego de estrategia, si bien difiere de este en muchos aspectos. Las reglas del Go son muy sencillas y, a pesar de que como en el Ajedrez, los jugadores enfrentan sus respectivas capacidades analíticas, en el Go la intuición juega un papel mucho más importante.
Un proverbio dice que nunca se ha jugado dos veces una misma partida de Go. Esto es bastante probable: sobre un tablero de 19 intersecciones por 19, hay alrededor de $4{,}63\times10^{170}$ posiciones posibles.

Se estima que es jugado por unos ochenta millones de personas en el sureste asiático, siendo los países predominantes China, Corea y Japón. En oriente normalmente se aprende a jugar en la infancia. Se juega al Go en las escuelas, en los comedores de las empresas, y en lugares especiales llamados "Salones de Go". Se juega por placer, para estimular la mente, o los jugadores profesionales por premios sustanciosos.

Aunque el juego del Go fue descrito por los viajeros europeos que fueron al Extremo Oriente durante el siglo XVII, no se empezó a jugar en Europa hasta 1880. Aun así el juego se ha ido extendiendo lentamente. En 1958 se jugó el primer campeonato de Europa. Hoy en día se juega al Go en todos los países europeos, si bien el nivel de juego está significativamente por debajo del de los profesionales orientales. Los mejores jugadores europeos acuden a Japón, China y Corea a estudiar el juego, convirtiéndose algunos en profesionales.

Es posible que fuera introducido en España por Sr. Ambrosio Wang An-po, ciudadano chino afincado en España, quien publicó un tratado de 181 páginas en castellano en 1970, titulado "El Cercado", del cual se han tomado algunos ejemplos. Tras el Sr. Wang An-Po, el Sr. Mikami, ha dedicado los últimos 20 años a la difusión del Go por todo el país, colaborando activamente en su promoción y en las relaciones de los jugadores españoles con los de otros países de oriente, especialmente Japón.

1.2 Características básicas del Go

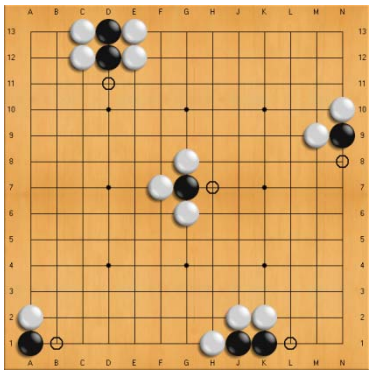
El Go se juega sobre un tablero que es una cuadrícula de 19 líneas verticales por 19 horizontales, formando 361 intersecciones. Los novatos suelen jugar en tableros más pequeños de 13×13 o de 9×9, sin otros cambios en las reglas de juego. Sobre las intersecciones se colocan alternativamente las piedras que son negras o blancas.

1.3 Las seis reglas básicas del Go

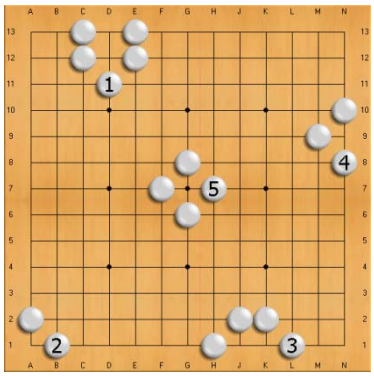
Para las imágenes de ejemplo utilizaremos un tablero de 9x9 y 13x13.

- Al inicio de la partida el tablero está vacío. Las Negras juegan primero. Después ambos jugadores juegan alternadamente. Una jugada consiste en poner una piedra en una intersección vacía.

- Cuando un jugador hace una jugada que priva de su última libertad a una piedra o formación del oponente debe sacar las piedras rodeadas del tablero y guardarlas separadamente hasta el final de la partida.

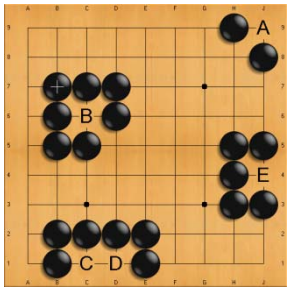


Negro está en atari

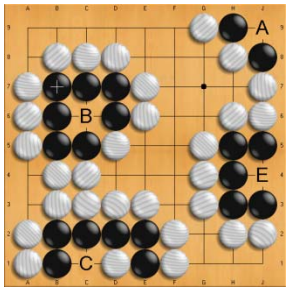


Blanco captura las piedras negras

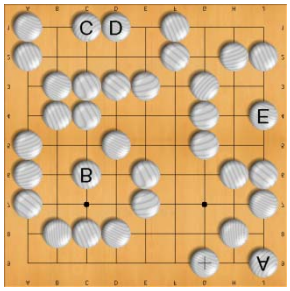
- No está permitido hacer una jugada ocupando la última libertad en el interior de una formación enemiga (suicidio) a no ser que esta jugada capture una o más piedras enemigas.



Blanco no puede mover a *"a"*, *"b"* o *"e"* pues sería suicidio.

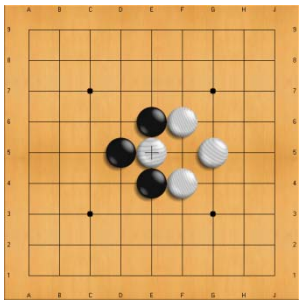
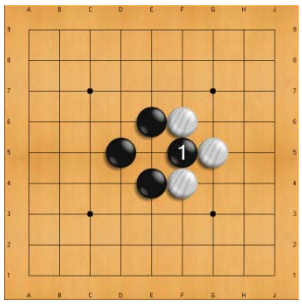


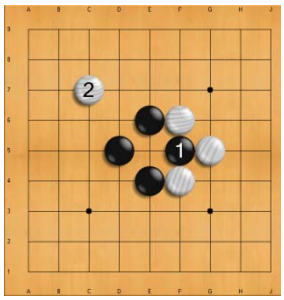
Blanco puede mover a *"a"*, *"b"* o *"e"* pues ha rodeado completamente al enemigo.



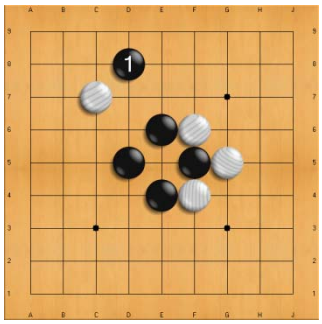
Blanco captura las piedras enemigas.

- Si un jugador captura una piedra en situación de ko, el otro jugador no puede recapturar inmediatamente. Ha de hacer otra jugada antes de recapturar.

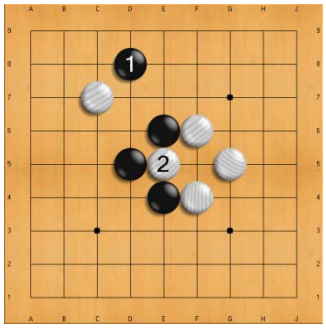





Situación de 'ko'. Las Blancas no pueden capturar en este turno a la piedra negra que destruyó su ojo. En vez de eso deben jugar en otro lugar.



Las Negras eligen no rellenar su ojo para defenderlo.



Consumido su turno de espera las Blancas pueden ya capturar la piedra negra. La posición final es una nueva situación de 'ko'

- En lugar de poner una piedra, un jugador puede pasar (perder un turno). Cuando los dos jugadores pasan consecutivamente, se acaba la partida.

- Una vez finalizada la partida se llenan todos los puntos neutrales. Después se retiran todas las piedras muertas y se añaden a las capturadas durante la partida. Ahora cada jugador recibe un punto por cada intersección vacía dentro de su propio territorio. A esto hay que añadir el número de prisioneros. El ganador es el que consigue el mayor número de puntos. Si los dos jugadores tienen el mismo número de puntos gana el Blanco como compensación por no jugar primero.

1.4 Estrategias básicas del Go

Aspectos básicos a tener en cuenta son:

- Conexión: tratar de mantener las piedras propias conectadas, así son más fáciles de defender.
- Corte: tratar de cortar las piedras del rival, así éste necesitará concentrarse en la defensa de más grupos.
- Vida: es la capacidad de las piedras para evitar ser eliminadas del tablero. Normalmente requiere la creación de grupos de piedras con al menos dos ojos.
- Muerte: ausencia de vida, tiene como resultado la eliminación de las piedras del tablero.

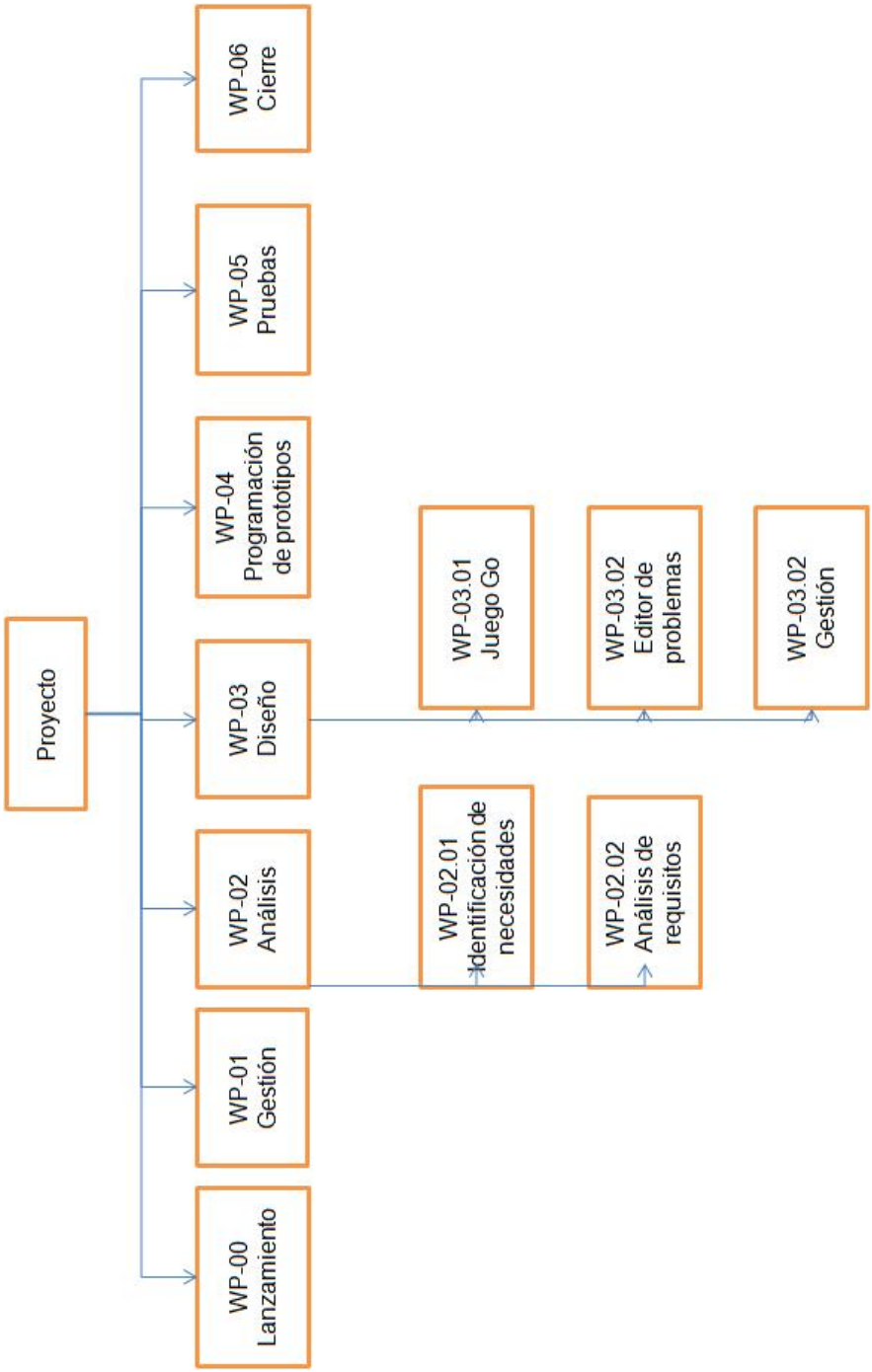
2. Objetivos del proyecto

La aplicación pretende ser un entorno en el cual los usuarios puedan aprender a jugar al Go a la vez que se divierten, procurar un entorno amigable para la creación y ejecución de problemas del juego y proporcionar de forma sencilla una manera de intercambiar los problemas creados entre los usuarios de la aplicación.

Los objetivos del proyecto son los siguientes:

- Desarrollar un interfaz agradable y cómoda tanto para el juego como para la creación de problemas.
- Desarrollar algoritmos para aplicar las reglas del juego.
- Desarrollar un sistema experto que implemente las estrategias básicas del juego.
- Desarrollar el proyecto cumpliendo la planificación especificada.
- Gestionar el borrado, la creación, la exportación y la importación de los problemas.
- Gestionar la creación y el borrado de los perfiles de los usuarios.

2.1 Estructura de la división del trabajo (EDT).



A continuación se realizará una breve descripción las actividades mostradas en el EDT:

- Lanzamiento (WP-00):

  En esta actividad se acordarán los términos en los que el proyecto debe ser realizado. Como salida de esta actividad se obtiene el Anexo B.

- Gestión (WP-01):

  Esta actividad se realizará a lo largo de todo el proyecto, consiste en comprobar que se cumple con la planificación obtenida y que se cumplen los objetivos propuestos, además de realizar la documentación necesaria del proyecto.

- Identificación de necesidades (WP-02.01):

  El objetivo de esta actividad es conocer las necesidades del cliente, para ello se concertarán entrevistas con el cliente.

  Como salida de esta actividad se redactará un informe con las conclusiones obtenidas.

- Análisis de requisitos (WP-02.02):

  En esta actividad se realizarán las siguientes actividades:

  - Reconocimiento del problema
  - Creación de la lista de requisitos
  - Casos de uso
  - Estudio del modelo conceptual de datos

  Una vez realizada esta actividad se obtendrán el modelo lógico del sistema, el modelo conceptual de datos y una lista de requisitos del sistema.

- Diseño del juego Go(WP-03.01):

  En esta actividad se establecerá la forma en la que interactúa el usuario con el entorno dedicado al juego, para ello se utilizarán los casos de uso, también se establecerán los distintos objetos, sus funcionalidades y sus relaciones, teniendo en cuenta las reglas y las estrategias básicas del juego, para ello crearemos diagramas de clase.

  En esta actividad diseñaremos también el aspecto gráfico del juego, y los distintos menús que necesitaremos para jugar.

  Y por último se establecerán las relaciones entre los distintos objetos para obtener las reglas y las estrategias básicas del juego.

  Como salida de esta actividad se obtendrán los diagramas de clase y el diseño del interfaz gráfico del juego.

- Diseño del editor de problemas (WP-03.01):

  En esta actividad se establecerá la forma en la que interactúa el usuario con el entorno dedicado a la creación de problemas, para ello se utilizarán los casos de uso, también se establecerán los distintos objetos, sus funcionalidades y sus relaciones, teniendo en cuenta las reglas básicas del juego necesarias para la edición de problemas, para ello crearemos diagramas de clase.

  En esta actividad diseñaremos también el aspecto gráfico del entorno dedicado a la creación de problemas, y los distintos menús que necesitaremos para la creación de problemas.

  Como salida de esta actividad se obtendrán los diagramas de clase y el diseño del interfaz gráfico del editor de problemas.

- Diseño de Gestión(WP-03.01):

  En esta actividad se establecerá la forma en la que interactúa el usuario con el entorno para la gestión de perfiles de usuarios y de problemas, para ello se utilizarán los casos de uso, también se establecerán los distintos objetos, sus funcionalidades y sus relaciones, que se usarán en la actividad de programación, para ello crearemos diagramas de clase.

  Y por último se realizará un diseño lógico de la base de datos para la gestión de problemas y perfiles de usuarios.

  Como salida de esta actividad se obtendrán los diagramas de clase y el diseño del interfaz gráfico para el entorno de gestión.

- Programación prototipos (WP-04):

  En esta actividad se programará el interfaz del juego, las reglas y se implementaran las estrategias del juego utilizando para ello el modelo lógico, los casos de uso y los diagramas de clase realizados en la actividad de análisis de la aplicación del juego Go. Esta actividad está basada en el desarrollo de distintos prototipos para alcanzar la aplicación final del juego.

  Como salida de esta actividad se obtendrá la aplicación final del juego Go.

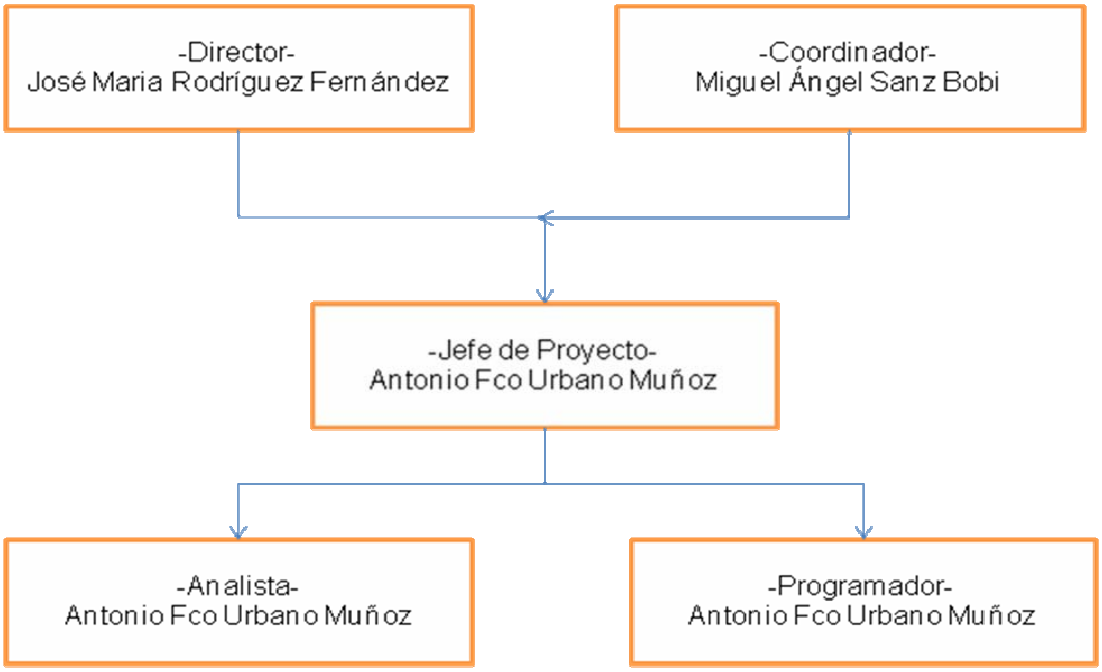
- Pruebas (WP-05):

  En esta actividad se comprobará que no se ha cometido ningún fallo en la programación, para ello se realizará una batería de pruebas en el ejecutable del prototipo de la aplicación.

- Cierre (WP-06):

  Una vez que se ha establecido que el prototipo es definitivo, se realizará un cierre formal del proyecto en la que se obtiene la aceptación de la aplicación por parte del cliente.

2.2 Organización



- Director: José María Rodríguez Fernández, es quien propuso el proyecto ”Desarrollo de un programa inteligente de aprendizaje y evaluación autónoma y competitiva del juego de estrategia Go” y es el encargado de garantizar la realización del proyecto, la calidad de sus contenidos, y que se desarrolla de acuerdo con las presentes normas, consultando con el Coordinador de Proyectos cuantas dudas le surjan.

- Coordinador: Miguel Ángel Sanz Bobi, es responsable de la evaluación del proyecto y de supervisar el trabajo del jefe del proyecto con el director del proyecto.

- Jefe de proyecto: Antonio Francisco Urbano Muñoz, es el encargado de la supervisión del analista y del programador, y de las actividades de lanzamiento, gestión, evaluación y cierre. También tiene que gestionar de forma eficiente el proyecto, y garantizar la calidad de éste.

- Analista: Antonio Fco Urbano Muñoz, es el encargado de realizar las actividades de análisis y diseño.

- Programador: Antonio Fco Urbano Muñoz, es el encargado de realizar las actividades de programación y pruebas

Responsabilidades:

	Coordinador	**Director**	**Jefe de Proyecto**	**Analista**	**Programador**
Lanzamiento	Informado	Informado	Ejecuta		
Gestión		Informado	Ejecuta	Informado	Informado
Análisis			Gestiona	Ejecuta	
Diseño			Gestiona	Ejecuta	Informado
Programación			Gestiona		Ejecuta
Pruebas			Gestiona		Ejecuta
Evaluación		Informado	Ejecuta		
Cierre	Informado	Informado	Ejecuta		

3. Diseño de la aplicación

3.1 Metodologías utilizadas [WIKI07]

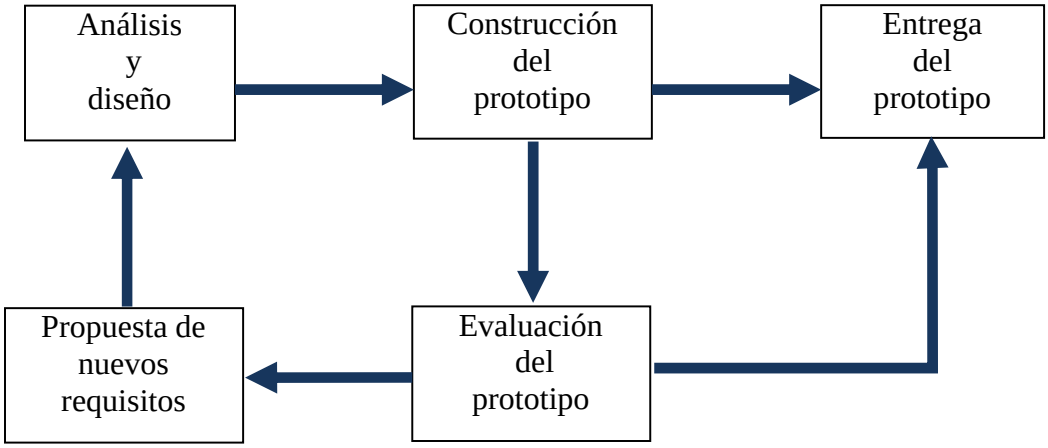
Se utilizará el modelo de desarrollo evolutivo (algunas veces denominado como prototipado evolutivo) construye una serie de grandes versiones sucesivas de un producto.

En el modelo evolutivo, los requerimientos son cuidadosamente examinados, y sólo esos que son bien comprendidos son seleccionados para el primer incremento. Los desarrolladores construyen una implementación parcial del sistema que recibe sólo estos requerimientos.

El sistema es entonces desarrollado, los usuarios lo usan, y proveen retroalimentación a los desarrolladores. Basada en esta retroalimentación, la especificación de requerimientos es actualizada, y una segunda versión del producto es desarrollada y desplegada. El proceso se repite indefinidamente.

El desarrollo evolutivo es 100% compatible con el modelo cascada. El desarrollo evolutivo no demanda una forma específica de observar el desarrollo de algún incremento. Así, el modelo cascada puede ser usado para administrar cada esfuerzo de desarrollo.

Las etapas del ciclo de vida a seguir son:



Las técnicas UML que se van a ser utilizadas son los casos de uso y diagramas de clase.

Los casos de uso describen lo que hace el sistema, no como lo hace. En el caso de uso se describe la interacción de los usuarios con el sistema.

Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas informáticos, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro.

Por lo que, con los casos de uso se establece una visión externa del desarrollo del proyecto, y con los diagramas de clase se define la funcionalidad de los objetos que compondrán la aplicación y sus relaciones.

El modelo que se va a seguir para la gestión de la base de datos, será el modelo relacional. En este modelo, el lugar y la forma en que se almacenen los datos no tienen relevancia (a diferencia de otros modelos como el jerárquico y el de red). Esto tiene la considerable ventaja de que es más fácil de entender y de utilizar para un usuario casual de la base de datos.

La información puede ser recuperada o almacenada por medio de «consultas» que ofrecen una amplia flexibilidad y poder para administrar la información.

Para ello utilizaremos una base de datos relacional, es un conjunto de dos o más tablas estructuradas en registros (líneas) y campos (columnas), que se vinculan entre sí por un campo en común, en ambos casos posee las mismas características como por ejemplo el nombre de campo, tipo y longitud; a este campo se le denomina ID, identificador o clave.

3.2 Identificación de necesidades

En esta etapa se define el problema a resolver y se fijan las normas a seguir para la dirección del proyecto, para ello se recoge información y a partir de ella se desarrolla el Documento de Conceptos del Sistema.
El cliente debe establecer sus objetivos y necesidades generales, también debe tener los conocimientos suficientes de la problemática a analizar para que estos objetivos y necesidades queden correctamente definidos y así el sistema final se ajuste totalmente a sus necesidades.

El documento de conceptos del sistema que se va a desarrollar contiene:
Objetivos de la aplicación: estos objetivos son desde un punto de vista empresarial, nunca informático.
Alcance de la aplicación: se trata de las funciones de negocio que se deben considerar dentro del alcance del proyecto.

3.2.1 Objetivos de la aplicación

Se pretende introducir la aplicación en el mercado de juegos de estrategia, creando así un gran impacto en el sector debido a que no existen juegos del Go como el que se va a desarrollar.

3.2.2 Alcance de la aplicación

Para la consecución de los objetivos establecidos la aplicación debe ser atractiva para el usuario y debe incluir un editor de problema y la forma de compartirlos entre los usuarios.

3.3 Análisis de requisitos [BARR01]

A continuación se realizarán las siguientes actividades:

- Reconocimiento del problema
- Creación de la lista de requisitos
- Casos de uso
- Estudio del modelo conceptual de datos

3.3.1 Reconocimiento del problema

Para entender suficientemente la aplicación que se pretende desarrollar se va a establecer brevemente los elementos que debe contener:

- Aplicar reglas de juego.
- Aplicar estrategias de juego.
- Aplicar reglas para la creación de problemas.
- Gestionar los problemas.
- Gestionar los perfiles de los usuarios.
- Diseñar un interfaz de juego amigable, intuitiva y atractiva para el usuario.
- Diseñar un interfaz para la creación de problemas amigable, intuitiva y atractiva para el usuario.

3.3.2 Creación de la lista de requisitos

La lista de requisitos consiste en una relación de requisitos establecidos por el cliente en las distintas entrevistas realizadas.

A continuación detallaremos los requisitos obtenidos:

Nombre del requisito: Menú inicial
Descripción: Menú inicial con las opciones para jugar, crear problemas, borrar problemas, crear perfiles de usuario, borrar perfiles de usuario, exportar problemas, importar problemas y salir de la aplicación.

Nombre del requisito: Perfil de usuario
Descripción: Cuando se opte por jugar, el usuario podrá escoger su perfil de usuario. En el perfil de usuario se almacenara toda la información referente a los problemas jugados por el usuario y el nivel de éste. También se deben ofrecer la opción de borrar un perfil de usuario ya creado.

Nombre del requisito: Entorno de juego
Descripción: El entorno de juego debe ser amigable, intuitivo y atractivo para el usuario. Además en el entorno de juego se debe indicar la situación de la partida, para ello se indicarán los movimientos que se vayan realizando durante la partida y las piedras capturadas por cada jugador.

Nombre del requisito: Entorno para la creación de problemas
Descripción: El entorno para la creación de problemas debe ser amigable, intuitivo y atractivo para el usuario. Además en el entorno de creación de problemas se tendrá que habilitar todas las herramientas necesarias para la creación de problemas.

Nombre del requisito: Selección de problemas

Descripción: Se procurará un menú en el que se permitirá al usuario elegir el problema que desea resolver. En el interfaz de juego debe tener un área que te muestre los comentarios referentes al problema seleccionado.

Nombre del requisito: Tablero

Descripción: El tablero del entorno de juego debe tener un tamaño de 19x19

Nombre del requisito: Piedras

Descripción: El color estándar de las piedras son el blanco y el negro.

Nombre del requisito: Base de datos

Descripción: La información referente a los problemas y a los perfiles de usuario se almacenará en una base de datos.

3.3.3 Casos de uso [WIKI07]

A partir de la lista de requisitos y del conocimiento del problema se crean los casos de uso, es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización software. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico. Normalmente, en los casos de usos se evita el empleo de jergas técnicas, prefiriendo en su lugar un lenguaje más cercano al usuario final.

Definiciones básicas para los diagramas de casos de uso:

- Actores : Se puede definir un actor como el rol o función que asume una persona, sistema o entidad que interactúa con el sistema que estamos construyendo de la misma forma. Tiene la propiedad de ser externo al sistema. Hay que tener en cuenta que un usuario puede acceder al sistema como distintos actores.
- Casos de uso: Como se ha dicho antes, un caso de uso es una secuencia de interacciones entre un sistema y alguien o algo que usa alguno de sus servicios. Ese alguien o algo es el actor. Tienen una representación gráfica de óvalos

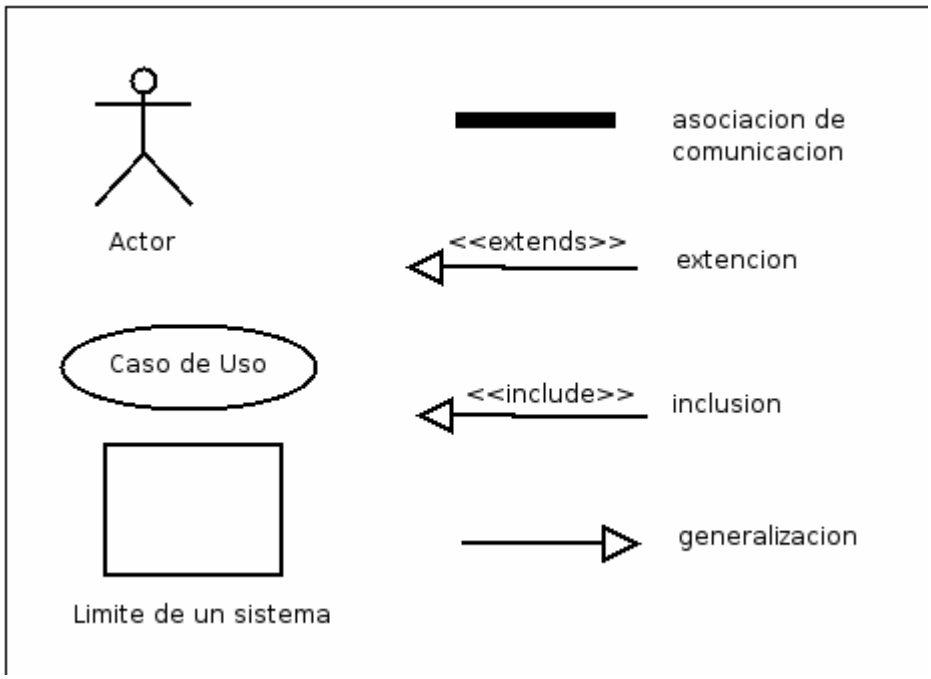
Como características principales, los casos de uso:

1. Están expresados desde el punto de vista del actor.
2. Se documentan con texto informal describiendo la interacción.
3. Están acotados al uso de una funcionalidad claramente definida del sistema o aplicación.

Situaciones que pueden darse:

- Un actor se comunica con un caso de uso.
- Un caso de uso extiende otro caso de uso.
- Un caso de uso usa otro caso de uso.

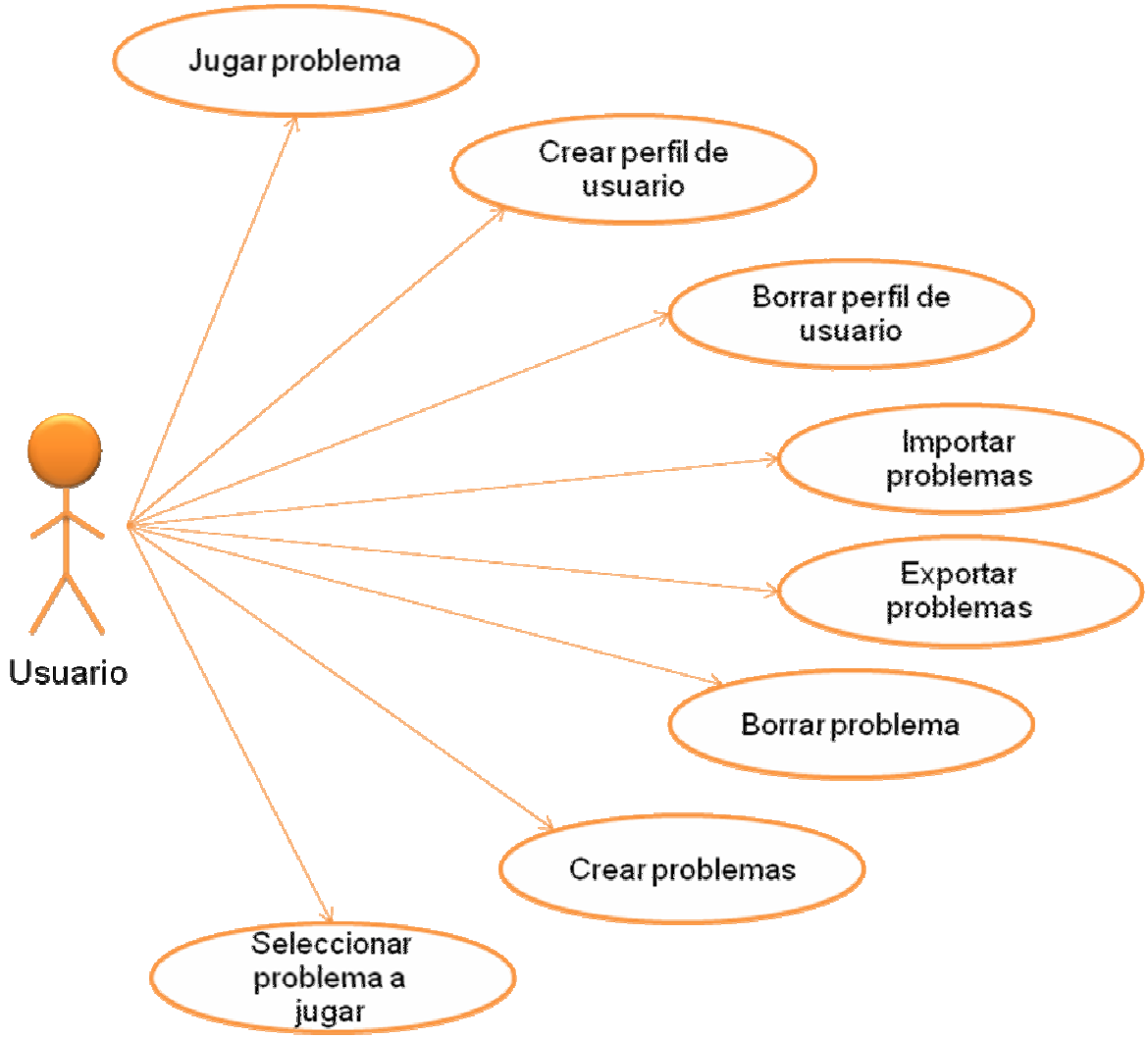
Notación de un caso de uso:



A continuación detallaremos los casos de uso existentes para la aplicación:

1. Jugar problema.
2. Crear perfil de usuario.
3. Borrar perfil de usuario.
4. Importar problemas.
5. Exportar problemas.
6. Crear problemas.
7. Seleccionar problema a jugar.
8. Borrar problema.

Diagrama de casos de uso del usuario:



Descripción detallada de casos de usos:

A continuación se describen los elementos que se van a utilizar para detallar los casos de uso:

Nombre: nombre del caso de uso.

Actor primario: aquél cuyo objetivo da nombre al caso de uso, normalmente será también el que lo inicie, pero esto no es siempre así. Suele participar en el caso de uso intercambiando información con el sistema. Siempre existe un actor primario en todo caso de uso.

Actores secundarios: pueden no existir, se tratan de cualquier otro actor que intervenga en el caso de uso y ayude al sistema a conseguir el objetivo del actor primario.

Precondiciones: condiciones que han de darse en el sistema para que pueda iniciarse el caso de uso. Puesto que se han de cumplir antes, no se vuelven a comprobar una vez iniciado el caso de uso.

Trigger: es el evento que inicia el caso de uso.

Escenario primario: pasos que se producen en el sistema para conseguir el objetivo del actor primario.

Extensiones: son formas alternativas de conseguir el objetivo.

Descripción de datos: en este apartado se describe la información intercambiada con el sistema.

Nombre	Jugar problema
Actor primario	Usuario
Actor secundario	-
Precondiciones	Haber seleccionado un problema
Trigger	Iniciado por el usuario
Escenario primario	1. El sistema muestra la jugada inicial. 2. El usuario realiza un movimiento. 3. El sistema comprueba que el movimiento finaliza una jugada correcta. 4. El sistema muestra el comentario de la jugada y actualiza el perfil del usuario.
Extensiones	2.a El movimiento no finaliza ninguna jugada. 2.a.1 Volvemos al punto 1. 2.b El movimiento finaliza una jugada que no es correcta. 2.b.1 El sistema muestra el comentario de la jugada.
Descripción de datos	Problema: - Idproblema - Titulo - Nivel - Jugada Inicial - Inicio - Comentario Jugadas: - Idjugada - Idproblema - Jugada - Comentario - Éxito Perfil: - Idperfil - Lvl - Problema

Nombre	Crear perfil de usuario
Actor primario	Usuario
Actor secundario	-
Precondiciones	-
Trigger	Iniciado por el usuario
Escenario primario	1. El usuario introduce su nombre de perfil de usuario 2. El sistema comprueba que el nombre de perfil de usuario no existe. 3. El sistema muestra añade el perfil de usuario al sistema.
Extensiones	2.a El nombre de perfil de usuario no existe. 2.a.1 Volvemos al punto 1.
Descripción de datos	Perfil: - Idperfil - Nombre - Lvl

Nombre	Borrar perfil de usuario
Actor primario	Usuario
Actor secundario	-
Precondiciones	-
Trigger	Iniciado por el usuario
Escenario primario	1. El sistema muestra todos los perfiles de usuarios. 2. El usuario selecciona el perfil de usuario a borrar 3. El sistema borra el perfil de usuario.
Extensiones	-
Descripción de datos	Perfil: - Idperfil - Lvl - Nombre

Nombre	Importar problema
Actor primario	Usuario
Actor secundario	-
Precondiciones	-
Trigger	Iniciado por el usuario
Escenario primario	1. El usuario selecciona el fichero a importar 2. El sistema comprueba que la extensión del fichero a importar es la correcta. 3. El sistema actualiza la información de los problemas.
Extensiones	2.a La extensión del fichero a importar no es la correcta 2.a.1 Volvemos al punto 1.
Descripción de datos	Problema: - Idproblema - Titulo - Nivel - Jugada Inicial - Inicio - Comentario Jugadas: - Idjugada - Idproblema - Jugada - Comentario - Éxito

Nombre	Exportar problema
Actor primario	Usuario
Actor secundario	-
Precondiciones	-
Trigger	Iniciado por el usuario
Escenario primario	1. El sistema muestra todos los problemas existentes. 2. El usuario selecciona el problema a exportar. 3. El sistema crea un fichero con la extensión .urb y exporta el problema.
Extensiones	-.
Descripción de datos	Problema: - Idproblema - Titulo - Nivel - Jugada Inicial - Inicio - Comentario Jugadas: - Idjugada - Idproblema - Jugada - Comentario - Éxito

Nombre	Borrar problema
Actor primario	Usuario
Actor secundario	-
Precondiciones	Haber seleccionado un problema
Trigger	Iniciado por el usuario
Escenario primario	1. El sistema muestra un listado con todos los problemas 2. El usuario selecciona el problema a borrar. 3. El sistema borra el problema de la base de datos.
Extensiones	-
Descripción de datos	Problema: - Idproblema - Titulo - Nivel - Jugada Inicial - Inicio - Comentario Jugadas: - Idjugada - Idproblema - Jugada - Comentario - Éxito

Nombre	Crear problema
Actor primario	Usuario
Actor secundario	-
Precondiciones	-
Trigger	Iniciado por el usuario
Escenario primario	1. El usuario introduce el nombre, creador y nivel del nuevo problema. 2. El sistema guarda la información del problema 3. El usuario introduce la jugada inicial del problema. 4. El sistema actualiza la información del problema. 5. El sistema muestra la jugada inicial del problema 6. El usuario introduce la jugada correcta del problema 7. El sistema actualiza la base de datos con la información del problema
Extensiones	6.a El usuario introduce una jugada que no es correcta 6.a.1 El sistema actualiza la base de datos con la información del problema. 6.a.2 Volvemos al punto 5.
Descripción de datos	Problema: - Idproblema - Titulo - Creador - Nivel - Jugada Inicial - Inicio - Comentario Jugadas: - Idjugada - Idproblema - Jugada - Comentario - Éxito

Nombre	Seleccionar problema a jugar
Actor primario	Usuario
Actor secundario	-
Precondiciones	-
Trigger	Iniciado por el usuario
Escenario primario	1. El sistema muestra un listado con todos los problemas 2. El usuario selecciona el problema. 3. El sistema muestra los datos del problema.
Extensiones	-
Descripción de datos	Problema: - Idproblema - Titulo - Creador - Nivel - Jugada Inicial - Inicio - Comentario

3.3.4 Modelo conceptual de datos [MARQ01]

Representa:

- Información lógica de un sistema
- Estructura
- Reglas

El diseño conceptual es un esquema o descripción de alto nivel de la estructura de los datos de un sistema independientemente de la implementación posterior de la base de datos.

Características:

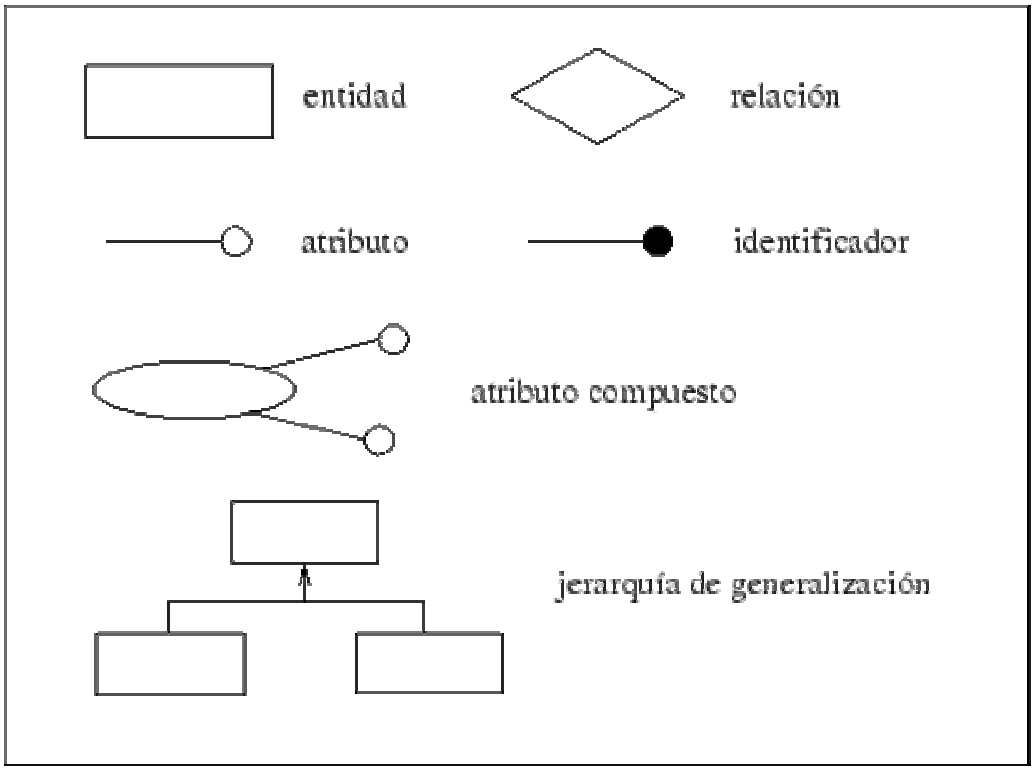
- Es un proceso dirigido completamente a los datos.
- Enfatiza la comprensión de los requerimientos de información del sistema.
- Permite una mejor comunicación entre usuarios, analistas y programadores durante todas las fases del diseño.
- Proporcionará las bases para diseñar una base de datos del sistema correcta, consistente, compartible y flexible.

Beneficios:

- Ayuda a valorar cuál es la tecnología óptima para desarrollar la base de datos del sistema.
- Permite reconocer como van a afectar posibles cambios en el futuro.
- Permite a los usuarios comprender cuáles serán sus datos en el sistema final sin tener este implementado.
- Favorece una visión del sistema y de cuáles son las necesidades reales de información.
- El modelo de datos facilita la migración de una base de datos a otra.

Para la representación grafica del modelo conceptual de datos utilizaremos el modelo entidad-relación.

El modelo entidad-relación es el modelo conceptual más utilizado para el diseño conceptual de bases de datos. El modelo entidad-relación está formado por un conjunto de conceptos que permiten describir la realidad mediante un conjunto de representaciones gráficas y lingüísticas.



Elementos del modelo entidad-relación:

- Relación: Es una correspondencia o asociación entre dos o más entidades. Cada relación tiene un nombre que describe su función. Las relaciones se representan gráficamente mediante rombos y su nombre aparece en el interior.

- Entidad: Cualquier tipo de objeto o concepto sobre el que se recoge información: cosa, persona, concepto abstracto o suceso. Por ejemplo: coches, casas, empleados, clientes, empresas, oficios, diseños de productos, conciertos, excursiones, etc. Las entidades se representan gráficamente mediante rectángulos y su nombre aparece en el interior. Un nombre de entidad sólo puede aparecer una vez en el esquema conceptual.

  Hay dos tipos de entidades: fuertes y débiles. Una entidad débil es una entidad cuya existencia depende de la existencia de otra entidad. Una entidad fuerte es una entidad que no es débil.

- Atributo: Es una característica de interés o un hecho sobre una entidad o sobre una relación. Los atributos representan las propiedades básicas de las entidades y de las relaciones. Toda la información extensiva es portada por los atributos. Gráficamente, se representan mediante bolitas que cuelgan de las entidades o relaciones a las que pertenecen.

  Cada atributo tiene un conjunto de valores asociados denominado dominio. El dominio define todos los valores posibles que puede tomar un atributo. Puede haber varios atributos definidos sobre un mismo dominio.

  Los atributos pueden ser simples o compuestos. Un atributo simple es un atributo que tiene un solo componente, que no se puede dividir en partes más pequeñas que tengan un significado propio. Un atributo compuesto es un atributo con varios componentes, cada uno con un significado por sí mismo. Un grupo de atributos se representa mediante un atributo compuesto cuando tienen afinidad en cuanto a su significado, o en cuanto a su uso. Un atributo compuesto se representa gráficamente mediante un óvalo.

Los símbolos que se van a utilizar para representar los diagramas entidad-relación son los siguientes:

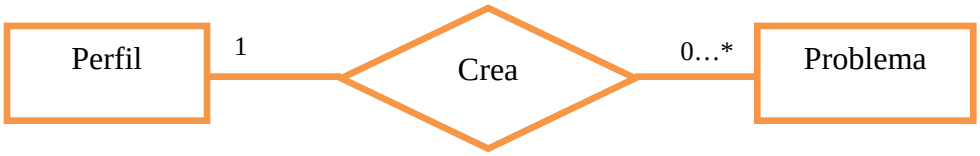
Para representar la cardinalidad de las relaciones utilizaremos los siguientes símbolos:

- 0,1 → Para la relación 0,1.
- 0 ... * → Para la relación de 0 a muchos.
- 1 …* → Para la relación de 1 a muchos.

Diagramas entidad-relación

El modelo conceptual de datos queda como se expone a continuación, siguiendo los símbolos especificados anteriormente:

Un perfil de usuario puede resolver ninguno o muchos problemas y un problema puede ser resuelto por ninguno o por muchos perfiles de usuarios.



Un perfil de usuario puede crear ninguno o muchos problemas y un problema es creado por un perfil de usuario.

Un perfil de usuario puede borrar ninguno o muchos problemas y un problema es borrado por un perfil de usuario.

Un perfil de usuario puede importar ninguno o muchos problemas y un problema es importado por un perfil de usuario.

Un perfil de usuario puede exportar ninguno o muchos problemas y un problema es exportado por un perfil de usuario.

Un perfil de usuario puede borrar ninguno o muchos perfiles y un perfil es borrado por un perfil de usuario.

Un perfil contiene ninguno o muchos problemas resueltos y un problema resuelto pertenece a ninguno o a muchos perfiles.

Un problema contiene una o muchas jugadas y una jugada pertenece a un problema.

Descripción de las entidades

Perfil de usuario: Contiene toda la información necesaria referente al usuario que está interactuando con el sistema.

Idperfil: Es el identificador del perfil.

Nombre: Es el nombre que el usuario le ha dado al perfil.

Lvl: Es el nivel que tiene el perfil de usuario.

Problema: Contiene toda la información necesaria para jugar un problema.

Idproblema: Es el identificador asignado al problema.

Título: Es el título del problema.

Creador: Almacena el nombre del creador del problema

Nivel: Indica el nivel a partir del cual se puede jugar el problema.

Jugada Inicial: Indica la secuencia de movimientos que establecen la jugada inicial o de partida para el problema.

Inicio: Indica que jugador debe iniciar los movimientos, si las piedras negras o las blancas.

Comentario: Contiene el comentario que ha introducido el creador del problema, se utiliza para indicar al usuario que vaya a jugar cual es el objetivo del problema.

Jugada: Contiene la información necesaria para la consecución de forma correcta de un problema.

Idjugada: Es el identificador de la jugada.

Idproblema: Indica el problema al que pertenece la jugada.

Jugada: Contiene la secuencia de movimientos que se debe seguir para realizar la jugada.

Comentario: Almacena los comentarios que indican si la jugada es correcto o una explicación de porque no debe realizarse esta jugada, los comentarios se muestran una vez finalizada la jugada.

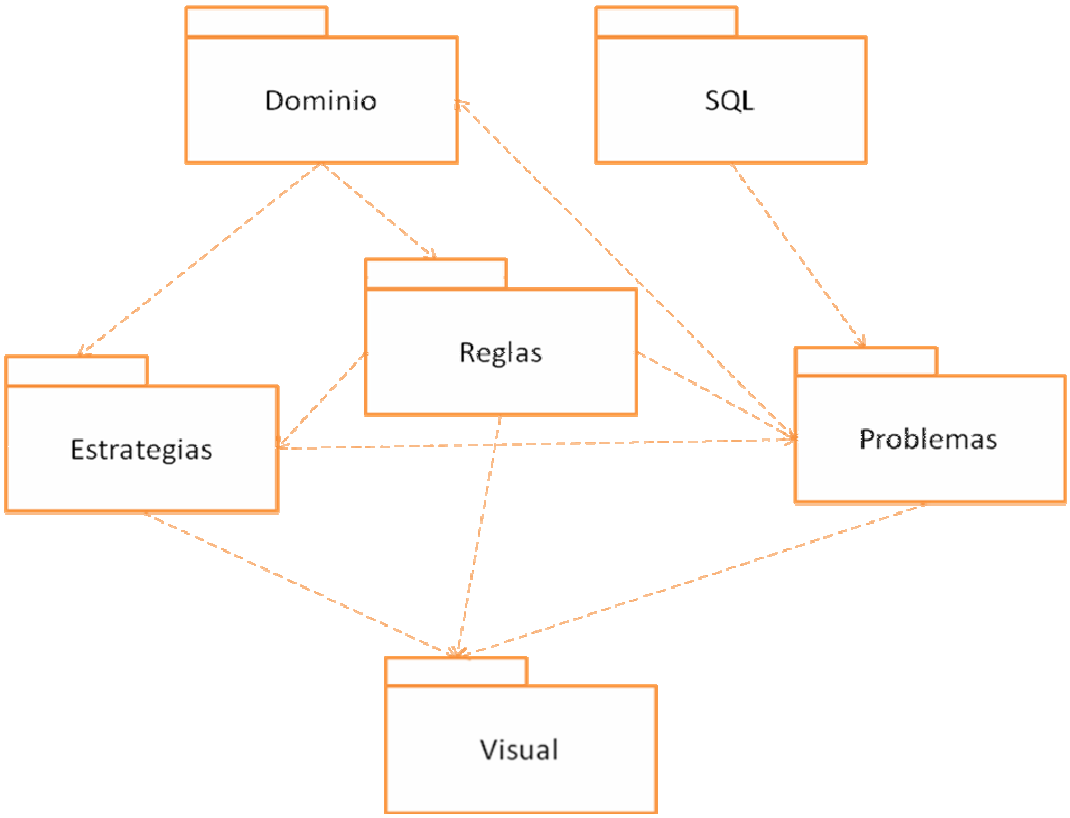
3.4 Diseño del juego Go

En este apartado definiremos tanto el diseño del interfaz gráfico del juego Go como el diseño lógico del mismo.

Diagrama de paquetes

Un diagrama de paquetes muestra como un sistema está dividido en agrupaciones lógicas mostrando las dependencias entre esas agrupaciones. Dado que normalmente un paquete está pensado como un directorio, los diagramas de paquetes suministran una descomposición de la jerarquía lógica de un sistema.

Los Paquetes están normalmente organizados para maximizar la coherencia interna dentro de cada paquete y minimizar el acoplamiento externo entre los paquetes. Con estas líneas maestras sobre la mesa, los paquetes son buenos elementos de gestión.



A continuación definiremos la función de los distintos paquetes lógicos que se van a utilizar para el desarrollo del juego Go, estos paquetes en realidad son módulos del juego Go:

Dominio: Este paquete contiene todo lo necesario para la definición de los elementos básicos del juego. A continuación definiremos los objetos más importantes que contienen este paquete:

- Pieza: Es el objeto básico de juego, contiene toda la información necesaria para definir una pieza, es decir, el color de la pieza, su ubicación y las libertades que posee. También contiene todos los métodos necesarios para interactuar con ella.
- Cadena: Es un conjunto de piezas conectadas. Una está conectada a otra cuando son del mismo color y las intersecciones en las que están posicionadas están conectadas por una línea. Por lo que una cadena solo puede ser de un color. Para definir una cadena necesitamos una colección de piezas conectadas y que sean todas del mismo color y las libertades de la cadena. También contiene todos los métodos necesarios para interactuar con ella.
- Perfil: Contiene toda la información necesaria del perfil de usuario que está en el entorno de juego. Es imprescindible para actualizar la información en la base de datos en caso de resolver un problema con éxito.

SQL: Este paquete contiene todos los objetos necesarios para interactuar con la base de datos, a continuación se señalaran los objetos más importantes que contiene:

- AccesoSql: Este objeto contiene todas las instrucciones necesarias para interactuar con las tablas de problemas, perfiles de usuario y jugadas tanto para lectura, creación, borrado o modificación de los registros que

contiene. También contiene todos los métodos necesarios para interactuar con él.

Problemas: Este paquete contiene todos los objetos necesarios para aplicar un problema con éxito, a continuación se señalaran los objetos más importantes que contiene:

- Jugada: Este objeto contiene todo lo necesario para la definición de una jugada perteneciente a un problema, para ello almacena la información de la jugada que se va a jugar. Esta información se compone de:
  - Identificador de la jugada.
  - Identificador del problema.
  - Secuencia de movimientos que representa la jugada.
  - Comentario de la jugada.
  - Un indicador de si la jugada es la correcta o no.

  Además contiene todos los métodos necesarios para interactuar con él.

- Problema: Este objeto contiene todo lo necesario para la definición de un problema, para ello almacena la información del problema que se va a jugar. Esta información se compone de:
  - Identificador del problema.
  - Jugada inicial del problema.
  - Comentario para el inicio del problema.
  - Color del jugador al comenzar el problema.
  - Una colección de las jugadas que componen el problema.

  Además contiene todos los métodos necesarios para interactuar con él.

- AplicarProblema: Este objeto contiene todo lo necesario para aplicar un problema, la estructura de este objeto se divide en tres partes. La primera establece el escenario inicial del juego, la segunda comprueba que existe una jugada en la que se está cumpliendo con la secuencia de movimientos, la tercera indica el siguiente movimiento a realizar. En la tercera parte continuamos la jugada en caso de encontrar alguna concordancia u obtenemos el siguiente movimiento que vamos a aplicar del módulo de estrategia. Además contiene todos los métodos necesarios para interactuar con él.

Reglas: Este paquete contiene todos los objetos necesarios para aplicar las reglas del juego con éxito, a continuación se señalaran los objetos más importantes que contiene:

- SituacionPartida: Este objeto contiene los algoritmos necesarios para conocer la situación de la partida. Además contiene todos los métodos necesarios para interactuar con él.
- Reglas: Este objeto contiene los algoritmos necesarios para aplicar las reglas de juego. Además contiene todos los métodos necesarios para interactuar con él.

Estrategias: Este paquete contiene todos los objetos necesarios para aplicar las estrategias básicas del juego con éxito, a continuación se señalaran los objetos más importantes que contiene:

- Estrategia: Este objeto contiene los algoritmos necesarios para aplicar las estrategias básicas del juego. Además contiene todos los métodos necesarios para interactuar con él.

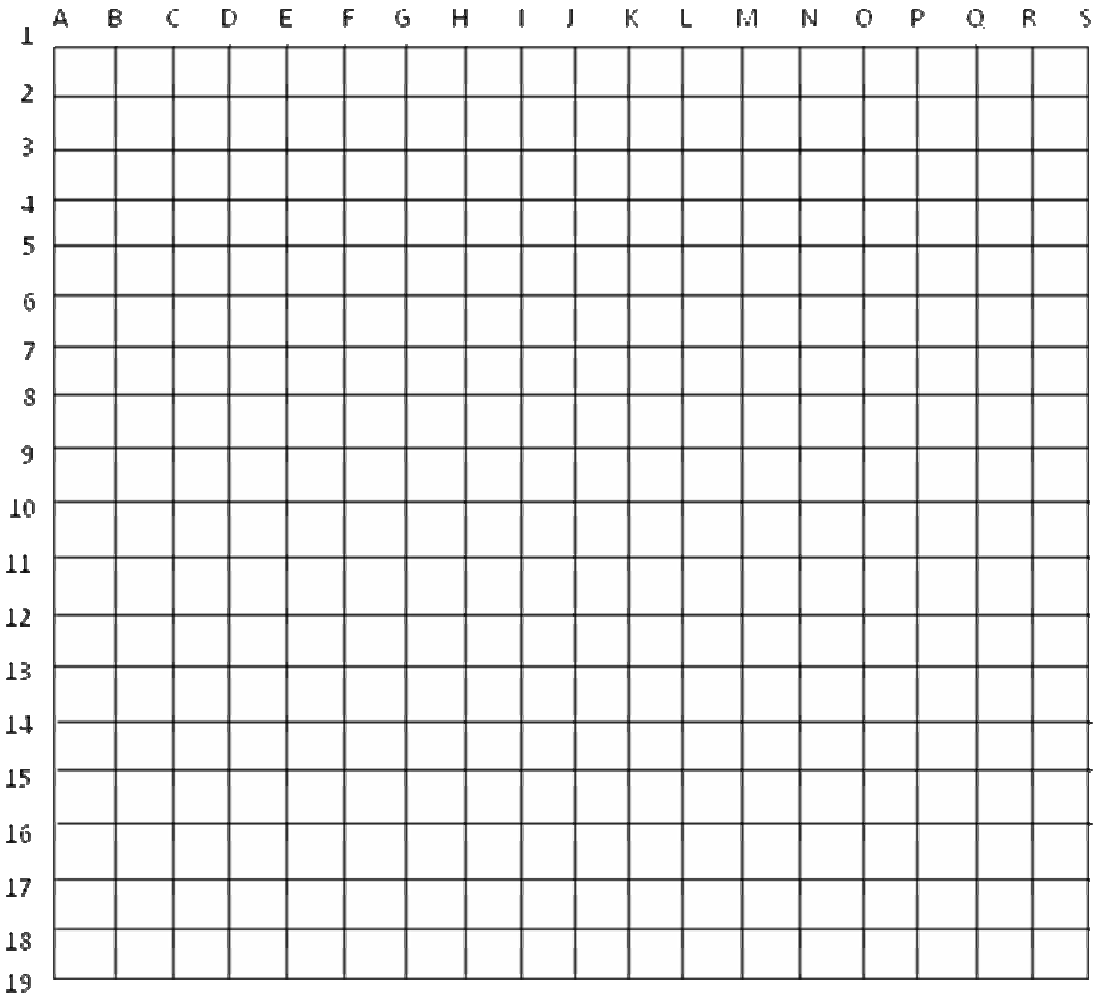
Visual: Este paquete contiene todos los objetos necesarios para mostrar el entorno en el que se va a desarrollar el juego, a continuación se señalaran los objetos más importantes que contiene:

- Go: Este objeto contiene los algoritmos necesarios para mostrar el tablero de juego. Además contiene todos los métodos necesarios para interactuar con él.
- Ventana: Este objeto contiene los algoritmos necesarios para mostrar el interfaz donde se va a desarrollar el juego, contiene a al objeto Go. Además contiene todos los métodos necesarios para interactuar con él.

Diseño de los elementos del interfaz de juego

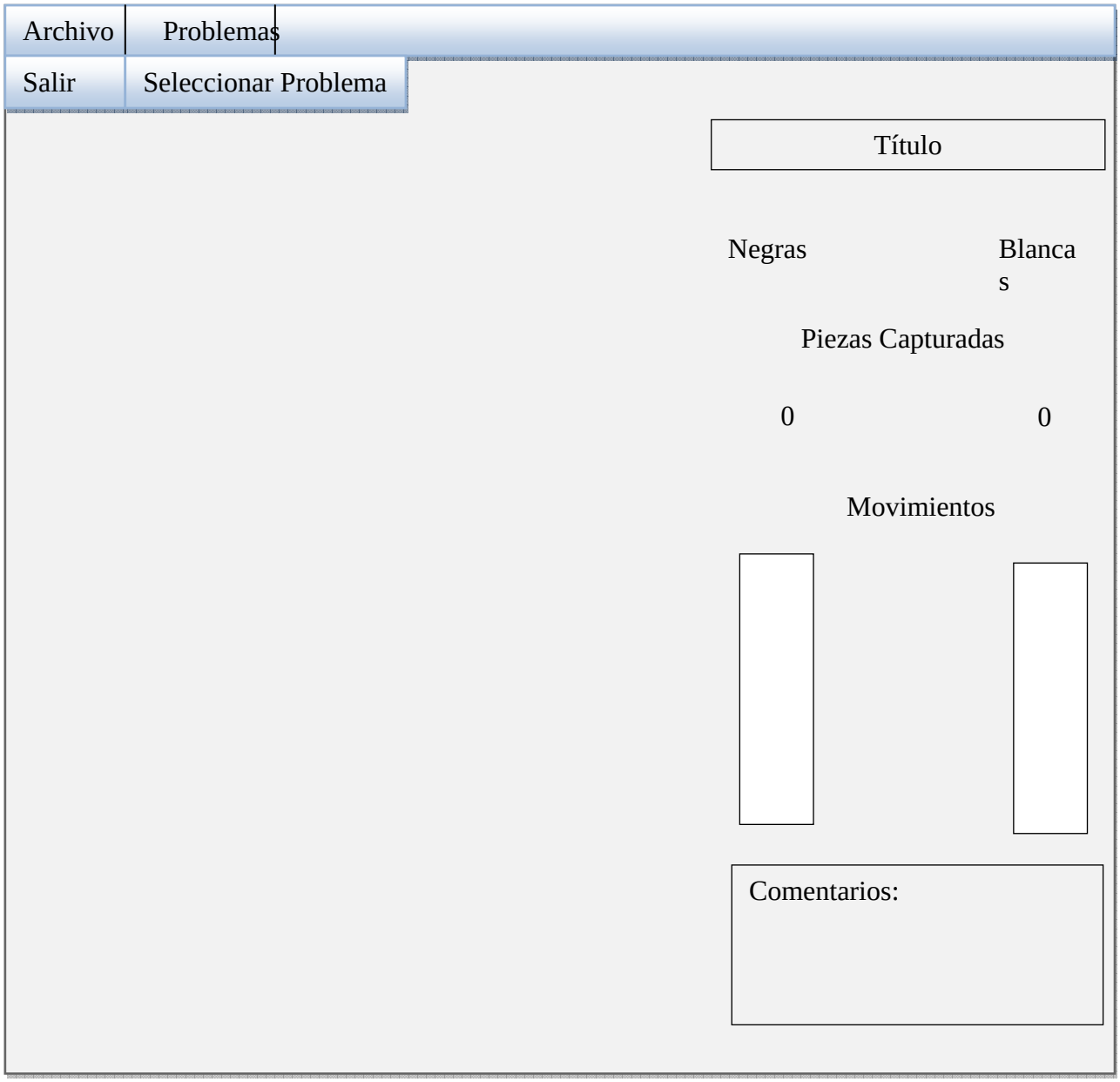
En este apartado mostraremos el diseño grafico que posteriormente programaremos para el entorno de juego.

- Tablero:



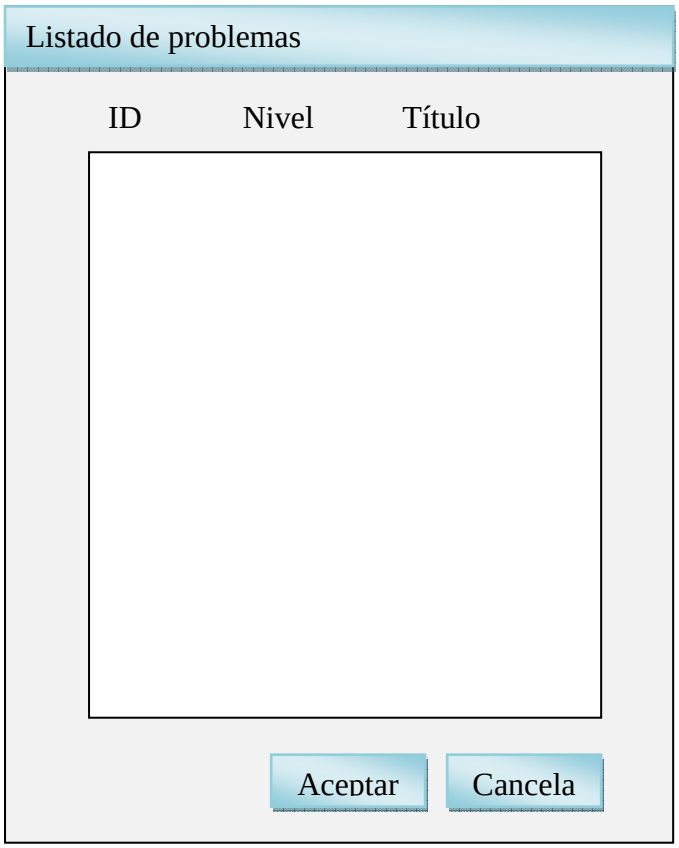
Este es el diseño que tendrá el tablero de juego, es un tablero de 19 x 19 donde el usuario al pinchar en las intersecciones podrá colocar las piedras.

➢ Ventana:



Esta será la disposición básica de los elementos en la ventana, en el hueco irá embebido el tablero de juego. Como se observa tenemos en el menú está la opción salir, que permitirá volver al menú inicial, se tendrá la opción de seleccionar problema, que conduce a otra ventana que se mostrará a continuación.

- Listado de problemas:

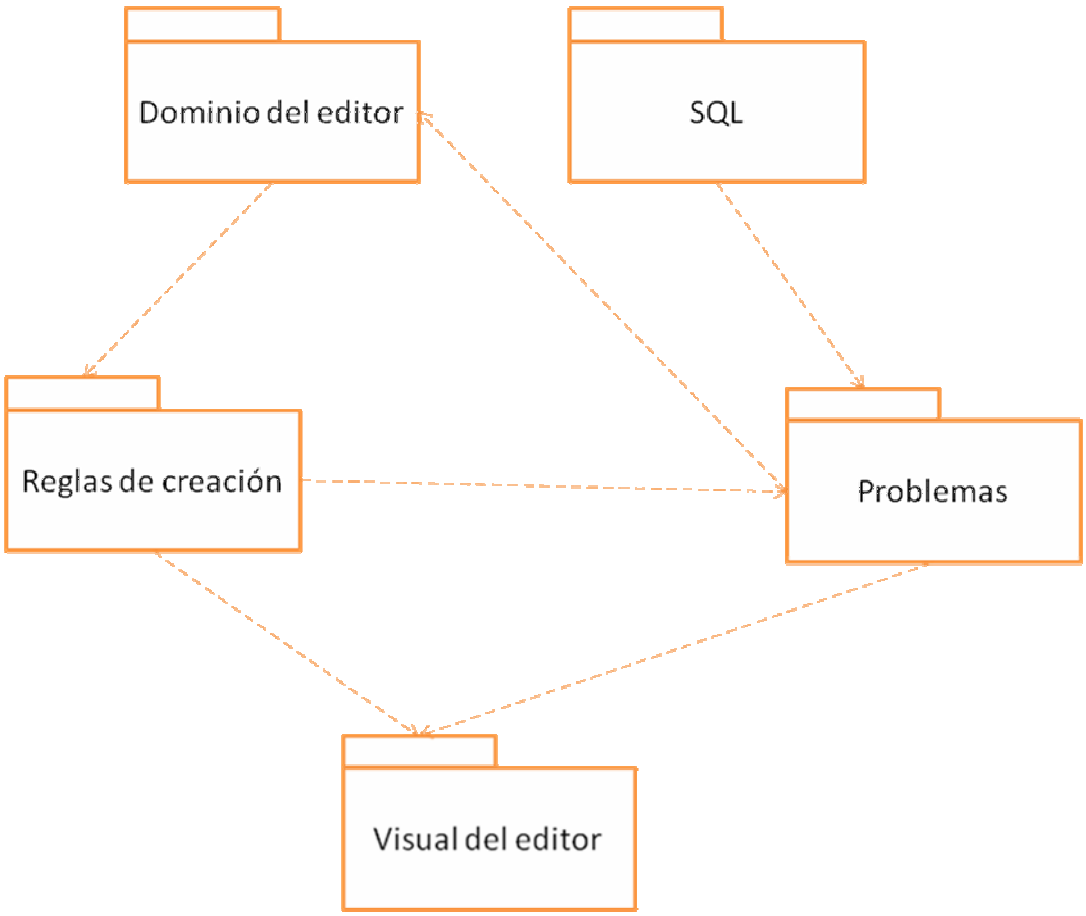


En esta ventana mostraremos un listado de los problemas seleccionable.

3.5 Diseño del editor de problemas

En este apartado definiremos tanto el diseño del interfaz gráfico del editor de problemas como el diseño lógico del mismo.

- Diagrama de paquetes



A continuación definiremos la función de los distintos paquetes lógicos que se van a utilizar para el desarrollo del juego Go, estos paquetes en realidad son módulos del juego Go:

Dominio del editor: Este paquete contiene todo lo necesario para la definición de los elementos básicos del juego. A continuación definiremos los objetos más importantes que contienen este paquete:

- Pieza: Es el objeto básico de juego, contiene toda la información necesaria para definir una pieza, es decir, el color de la pieza, su ubicación y las libertades que posee. También contiene todos los métodos necesarios para interactuar con ella.
- Cadena: Es un conjunto de piezas conectadas. Una está conectada a otra cuando son del mismo color y las intersecciones en las que están posicionadas están conectadas por una línea. Por lo que una cadena solo puede ser de un color. Para definir una cadena necesitamos una colección de piezas conectadas y que sean todas del mismo color y las libertades de la cadena. También contiene todos los métodos necesarios para interactuar con ella.

SQL: Este paquete contiene todos los objetos necesarios para interactuar con la base de datos, a continuación se señalaran los objetos más importantes que contiene:

- AccesoSql: Este objeto contiene todas las instrucciones necesarias para interactuar con las tablas de problemas, perfiles de usuario y jugadas tanto para lectura, creación, borrado o modificación de los registros que contiene. También contiene todos los métodos necesarios para interactuar con él.

Problemas: Este paquete contiene todos los objetos necesarios para crear un problema con éxito, a continuación se señalaran los objetos más importantes que contiene:

- Jugada: Este objeto contiene todo lo necesario para la definición de una jugada perteneciente a un problema, para ello almacena la información de la jugada que se va a jugar. Esta información se compone de:
  - Identificador de la jugada.
  - Identificador del problema.
  - Secuencia de movimientos que representa la jugada.
  - Comentario de la jugada.
  - Un indicador de si la jugada es la correcta o no.

Además contiene todos los métodos necesarios para interactuar con él.

- Problema: Este objeto contiene todo lo necesario para la definición de un problema, para ello almacena la información del problema que se va a jugar. Esta información se compone de:
  - Identificador del problema.
  - Título
  - Creador
  - Nivel
  - Jugada inicial del problema.
  - Comentario para el inicio del problema.
  - Color del jugador al comenzar el problema.
  - Una colección de las jugadas que componen el problema.

Además contiene todos los métodos necesarios para interactuar con él.

- CrearProblema: Este objeto contiene toda la lógica necesaria para la creación de un problema, es el nexo entre los distintos módulos de la aplicación. Además contiene todos los métodos necesarios para interactuar con él.

Reglas de creación: Este paquete contiene todos los objetos necesarios para aplicar las reglas del juego necesarias para la creación de un problema, estas reglas difieren un poco de las reglas de juego, a continuación se señalaran los objetos más importantes que contiene:

- SituacionPartida: Este objeto contiene los algoritmos necesarios para conocer la situación de la partida. Además contiene todos los métodos necesarios para interactuar con él.
- Reglas: Este objeto contiene los algoritmos necesarios para aplicar las reglas para la creación de problemas. Además contiene todos los métodos necesarios para interactuar con él.

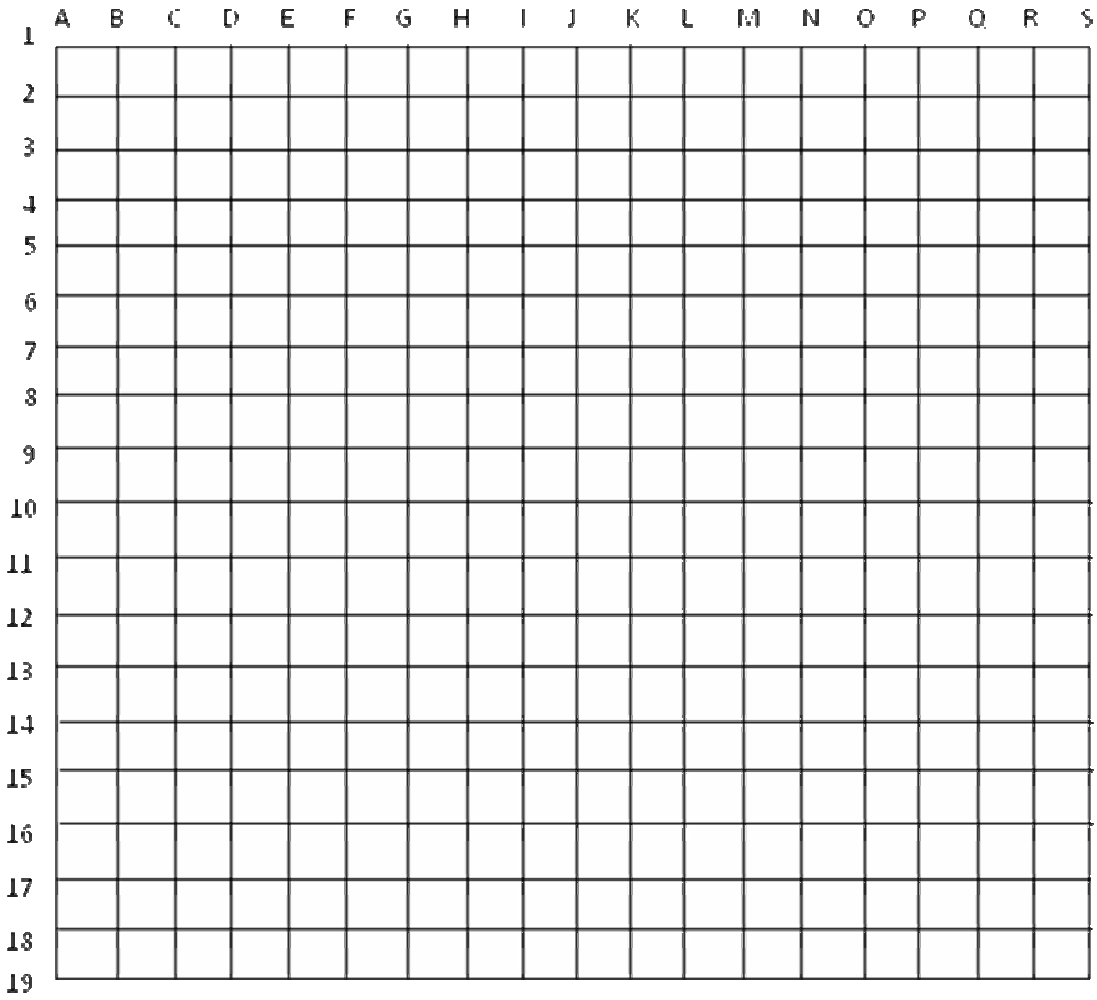
Visual del editor: Este paquete contiene todos los objetos necesarios para mostrar el entorno en el que se va a desarrollar el juego, a continuación se señalaran los objetos más importantes que contiene:

- Editor: Este objeto contiene los algoritmos necesarios para mostrar el tablero de juego. Además contiene todos los métodos necesarios para interactuar con él.
- Ventana del Editor: Este objeto contiene los algoritmos necesarios para mostrar el interfaz donde se va a desarrollar el juego, contiene a al objeto Go. Además contiene todos los métodos necesarios para interactuar con él.

Diseño de los elementos del interfaz del editor de problemas

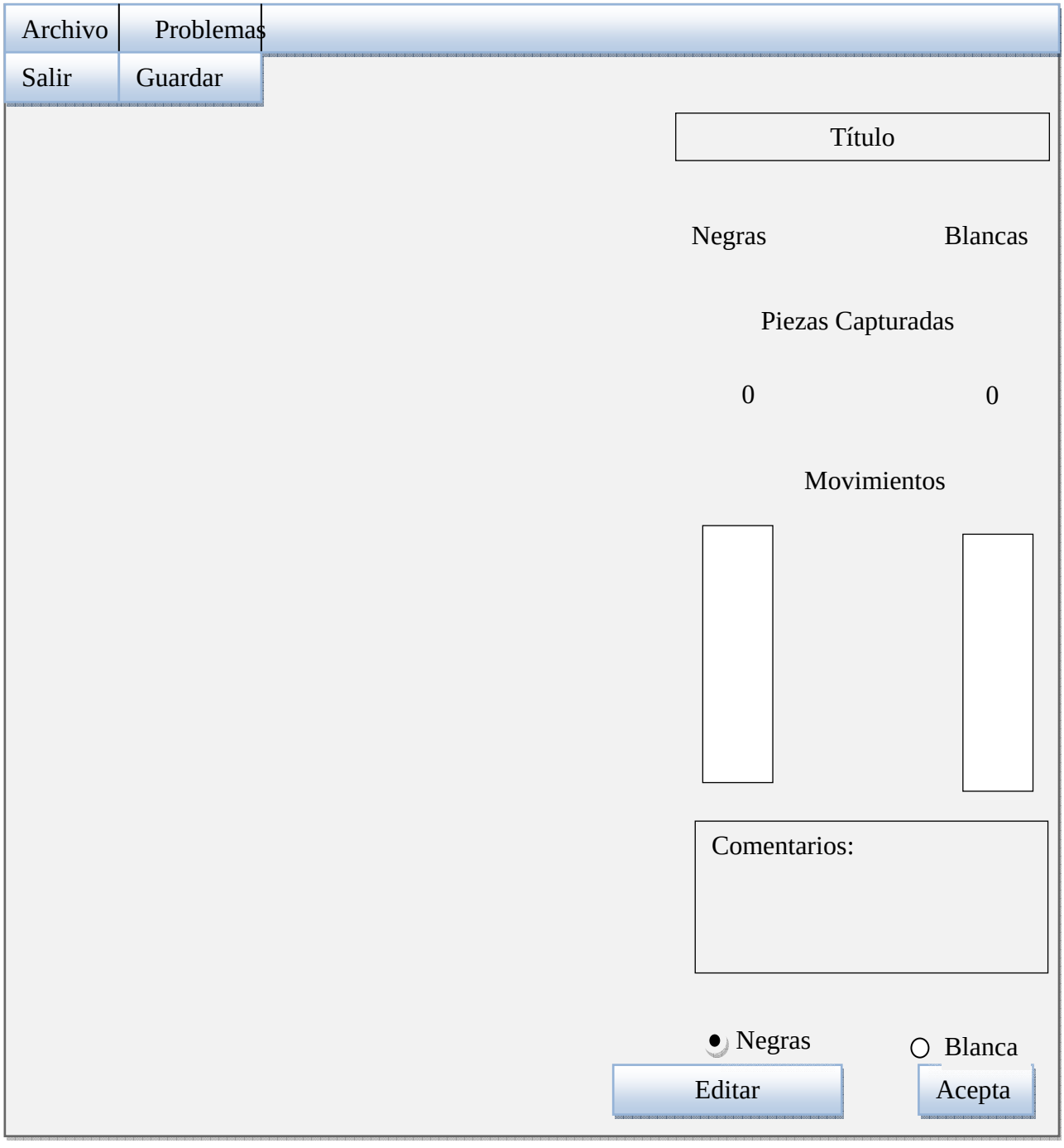
En este apartado mostraremos el diseño grafico que posteriormente programaremos para el editor de problemas.

- Tablero:



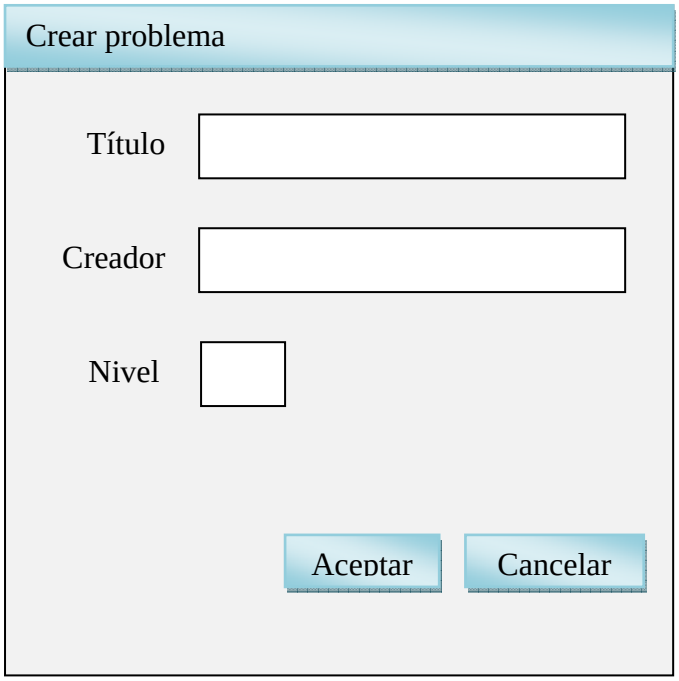
Este es el diseño que tendrá el tablero de juego, es un tablero de 19 x 19 donde el usuario al pinchar en las intersecciones podrá colocar las piedras.

➢ Ventana:



Esta será la disposición básica de los elementos en la ventana, en el hueco irá embebido el tablero de juego. Como se observa, en el menú está la opción salir, que permitirá volver al menú inicial, también está la opción de guardar problema, que almacenará el problema creado en la base de datos.

- Crear problema:

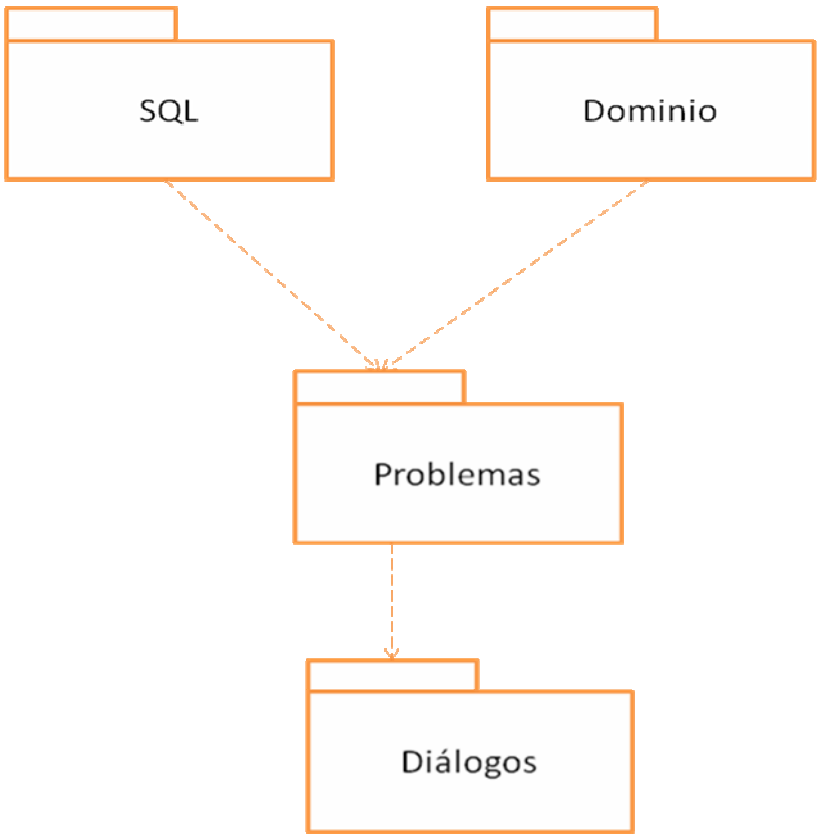


En esta ventana introduciremos la información necesaria para la creación de problemas.

3.6 Diseño de la gestión

En este apartado definiremos tanto el diseño del interfaz gráfico del editor de la gestión como el diseño lógico del mismo.

- Diagrama de paquetes



A continuación definiremos la función de los distintos paquetes lógicos que se van a utilizar para el desarrollo de la gestión:

Dominio: Este paquete contiene los elementos básicos para la creación de un problema y los elementos para la gestión de perfiles. A continuación definiremos los objetos más importantes que contienen este paquete:

- Pieza: Es el objeto básico de juego, contiene toda la información necesaria para definir una pieza, es decir, el color de la pieza, su ubicación y las libertades que posee. También contiene todos los métodos necesarios para interactuar con ella.
- Cadena: Es un conjunto de piezas conectadas. Una está conectada a otra cuando son del mismo color y las intersecciones en las que están posicionadas están conectadas por una línea. Por lo que una cadena solo puede ser de un color. Para definir una cadena necesitamos una colección de piezas conectadas y que sean todas del mismo color y las libertades de la cadena. También contiene todos los métodos necesarios para interactuar con ella.

- Perfil: Contiene toda la información necesaria del perfil de usuario que está en el entorno de juego. Es imprescindible para actualizar la información en la base de datos en caso de resolver un problema con éxito.

SQL: Este paquete contiene todos los objetos necesarios para interactuar con la base de datos, a continuación se señalaran los objetos más importantes que contiene:

- AccesoSql: Este objeto contiene todas las instrucciones necesarias para interactuar con las tablas de problemas, perfiles de usuario y jugadas tanto para lectura, creación, borrado o modificación de los registros que contiene. También contiene todos los métodos necesarios para interactuar con él.

Problemas: Este paquete contiene todos los objetos necesarios para crear un problema con éxito, a continuación se señalaran los objetos más importantes que contiene:

- Jugada: Este objeto contiene todo lo necesario para la definición de una jugada perteneciente a un problema, para ello almacena la información de la jugada que se va a jugar. Esta información se compone de:
  - Identificador de la jugada.
  - Identificador del problema.
  - Secuencia de movimientos que representa la jugada.
  - Comentario de la jugada.
  - Un indicador de si la jugada es la correcta o no.

  Además contiene todos los métodos necesarios para interactuar con él.

- Problema: Este objeto contiene todo lo necesario para la definición de un problema, para ello almacena la información del problema que se va a jugar. Esta información se compone de:
  - Identificador del problema.
  - Título
  - Creador
  - Nivel
  - Jugada inicial del problema.
  - Comentario para el inicio del problema.
  - Color del jugador al comenzar el problema.
  - Una colección de las jugadas que componen el problema.

  Además contiene todos los métodos necesarios para interactuar con él.

Diálogos: Este paquete contiene todos los diálogos necesarios para la gestión del juego, a continuación se señalaran los objetos más importantes que contiene:

- Inicio: Muestra las opciones de inicio del juego:
  - Jugar al Go
  - Editor de problemas
  - Borrar problema
  - Borrar perfil
  - Exportar problema
  - Importar problema
  - Salir

  Además contiene todos los métodos necesarios para interactuar con él.

- Borrar problema: Este diálogo muestra una lista de problemas seleccionable que pueden ser borrados. Además contiene todos los métodos necesarios para interactuar con él.

- Borrar perfil: Este diálogo muestra una lista de perfiles seleccionable que pueden ser borrados. Además contiene todos los métodos necesarios para interactuar con él.

- Exportar problema: Este diálogo muestra una lista de problemas seleccionable que pueden ser exportados a un fichero con el formato ".urb". Además contiene todos los métodos necesarios para interactuar con él.

- Importar problema: Este diálogo muestra raíz de directorios en la que podremos navegar para importar los problemas con el formato ".urb". Además contiene todos los métodos necesarios para interactuar con él.

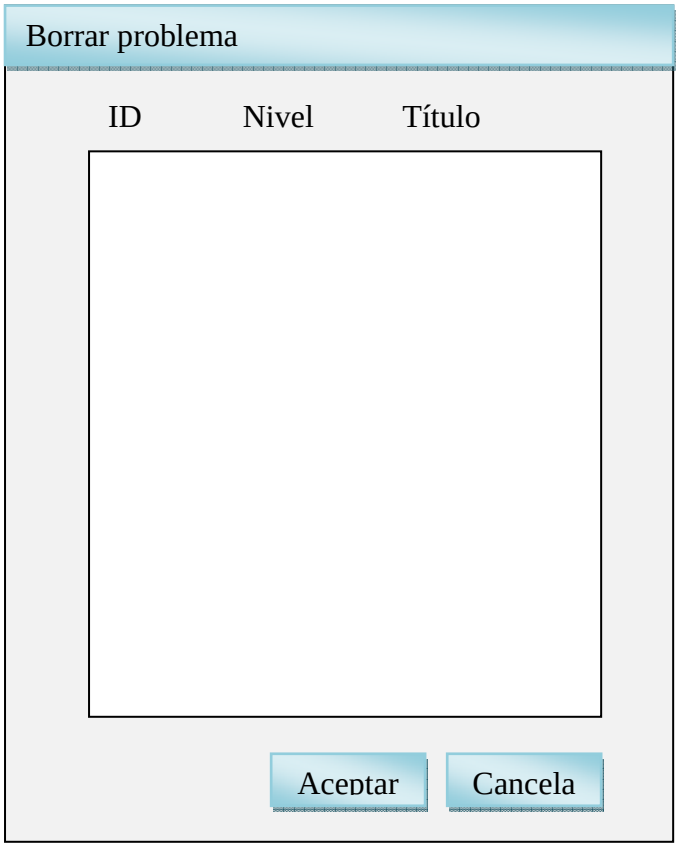
- Seleccionar perfil: Este diálogo muestra una lista de perfiles seleccionable, una vez seleccionado un perfil podremos subirlo de nivel resolviendo los problemas existentes en la base de datos. Además contiene todos los métodos necesarios para interactuar con él.

- Borrar perfil: Este diálogo muestra una lista de perfiles seleccionable, una vez seleccionado un perfil podremos borrarlo de la base de datos. Además contiene todos los métodos necesarios para interactuar con él.

Diseño de los elementos del interfaz de gestión

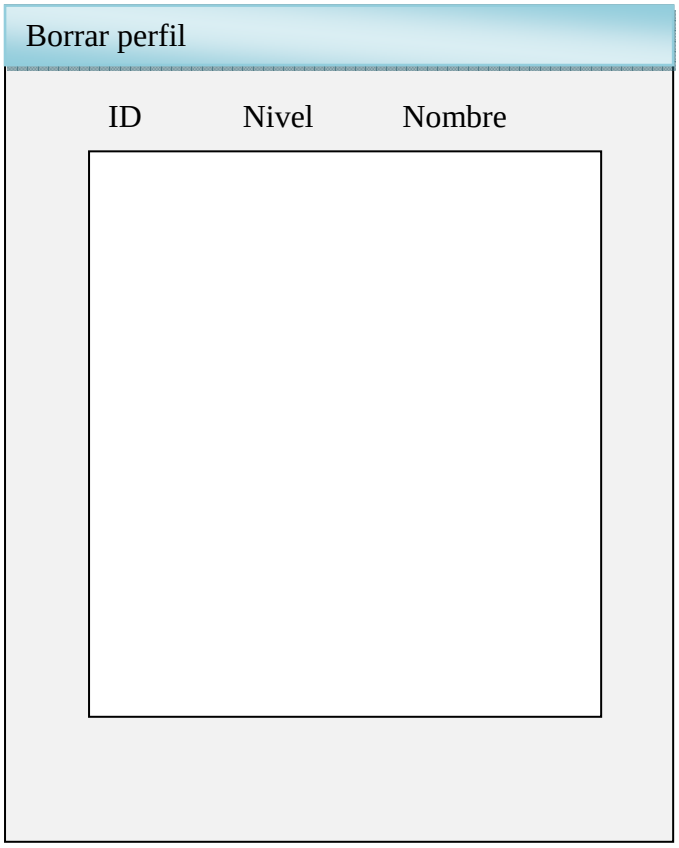
En este apartado mostraremos el diseño grafico que posteriormente programaremos para el entorno de gestión.

- Borrar problema:



En esta ventana se mostrará un listado seleccionable de los problemas, una vez seleccionado el problema si le damos a aceptar borraremos el problema de la base de datos.

- Borrar perfil:

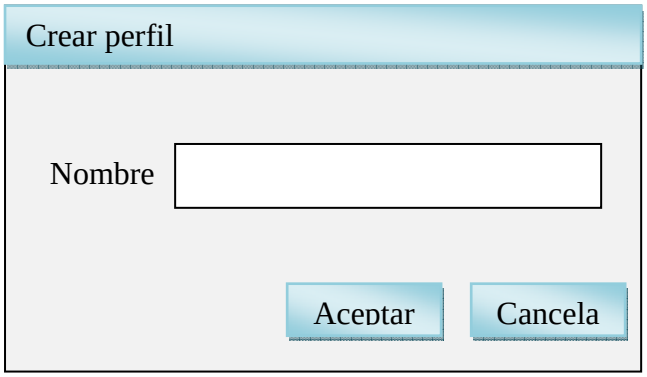


En esta ventana se mostrará un listado seleccionable de los perfiles, una vez seleccionado el perfil si le damos a aceptar borraremos el problema de la base de datos.

- Seleccionar perfil:

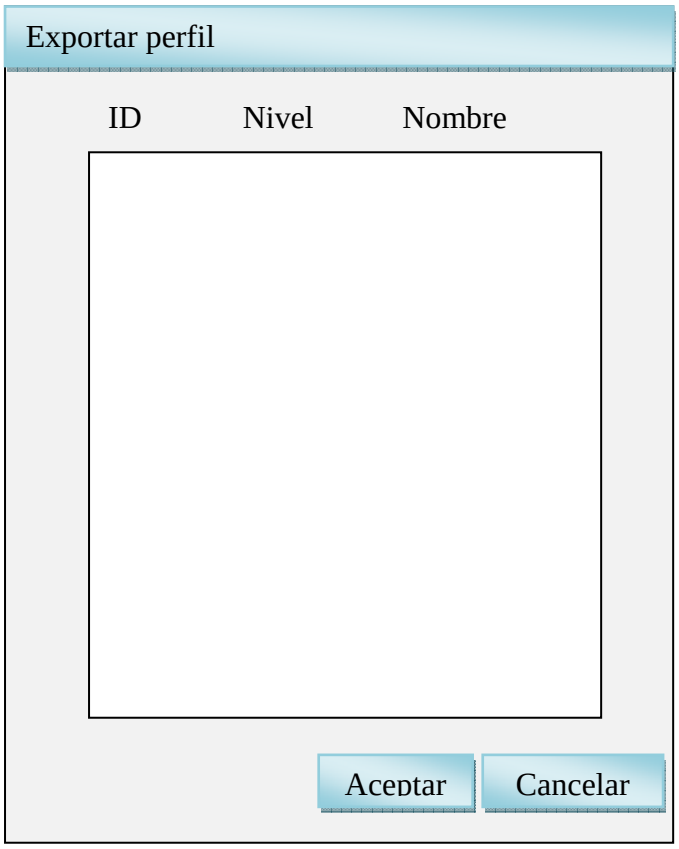
En esta ventana se mostrará un listado seleccionable de los perfiles, una vez seleccionado el perfil si le damos a aceptar comenzaremos a jugar con el perfil seleccionado.

- Crear perfil:



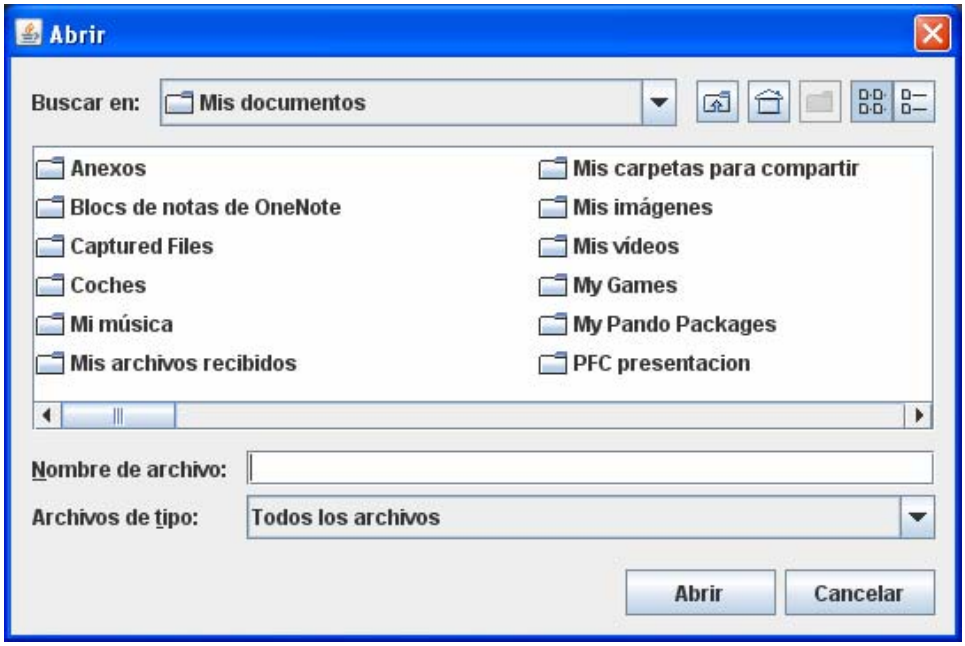
En esta ventana introduciremos la información necesaria para la creación de perfiles, actualizando la base de datos en el momento que aceptamos el nombre de perfil introducido.

- Exportar problema:



En esta ventana se mostrará un listado seleccionable de los problemas, una vez seleccionado el problema si le damos a aceptar exportaremos el problema con el formato ".urb" en la carpeta donde tengamos el ejecutable del juego.

- Importar problema



El aspecto de la ventana será concretamente este, debido a que en Java existe un diálogo para la selección de archivos con un aspecto predefinido. Se deberá restringir los archivos que se puedan importar a los archivos con la extensión ".urb".

3.7 Diseño de la base de datos

Para comprender mejor la estructura de los datos se va a definir las tablas y los atributos necesarios para almacenar la información imprescindible para el desarrollo del proyecto.

Para ello definiremos las distintas entidades necesarias, los atributos, y las relaciones entre las entidades, describiendo los siguientes puntos:

- Atributos: se describirán las características del elemento.
- Clave primaria: se definirá la clave primaria de la tabla.
- Clave extranjera: En caso de tener clave externa, se indicará en este apartado.
- Descripción: se describirá la tabla y los elementos que la componen.

Jugadas	
Clave primaria	Idjugada
Clave extranjera	Idproblema

Esta tabla almacena toda la información necesaria para definir una jugada, para ello necesitamos los siguientes elementos:

- Idjugada: Identifica unívocamente al registro de la jugada en cuestión.
- Idproblema: Identifica el problema al que pertenece la jugada.
- Jugada: Almacena los movimientos necesarios para la realización correcta de la jugada.
- Comentario: Contiene el texto que se mostrará cuando se realice la jugada.
- Éxito: Determina si la jugada es válida para dar el problema al que pertenece por resuelto.

Elementos	*Atributos*
Idjugada	Es un entero de 11, clave primaria. Es auto incremental. No se permiten nulos.
Idproblema	Es un entero de 11, clave extranjera. No se permiten nulos.
Jugada	Es un varchar de 2000 caracteres. Se permiten nulos.
Comentario	Es un varchar de 200 caracteres. Se permiten nulos.
Éxito	Es un entero de 1. No se permiten nulos.

Perfil	
Clave primaria	Idperfil
Clave extranjera	
Esta tabla almacena toda la información necesaria para definir un perfil, para ello necesitamos los siguientes elementos: • Idperfil: Identifica unívocamente al registro del perfil en cuestión. • Nombre: Contiene el nombre del perfil asignado por el usuario. • Lvl: Almacena el nivel del usuario.	
Elementos	*Atributos*
Idperfil	Es un entero de 11, clave primaria. Es auto incremental. No se permiten nulos.
Nombre	Es un varchar de 50 caracteres. No se permiten nulos.
Lvl	Es un entero de 11. No se permiten nulos.

Problemas-Perfil	
Clave primaria	Idperfil
Clave extranjera	Idproblema
Esta tabla almacena toda la información necesaria para conocer qué problemas han sido solucionados satisfactoriamente por un perfil, para ello necesitamos los siguientes elementos: • Idperfil: Identifica unívocamente al registro del perfil en cuestión. • Idproblema: Identifica unívocamente el problema solucionado.	
Elementos	*Atributos*
Idperfil	Es un entero de 11, clave primaria. No se permiten nulos.
Idproblema	Es un entero de 11, clave primaria. No se permiten nulos.

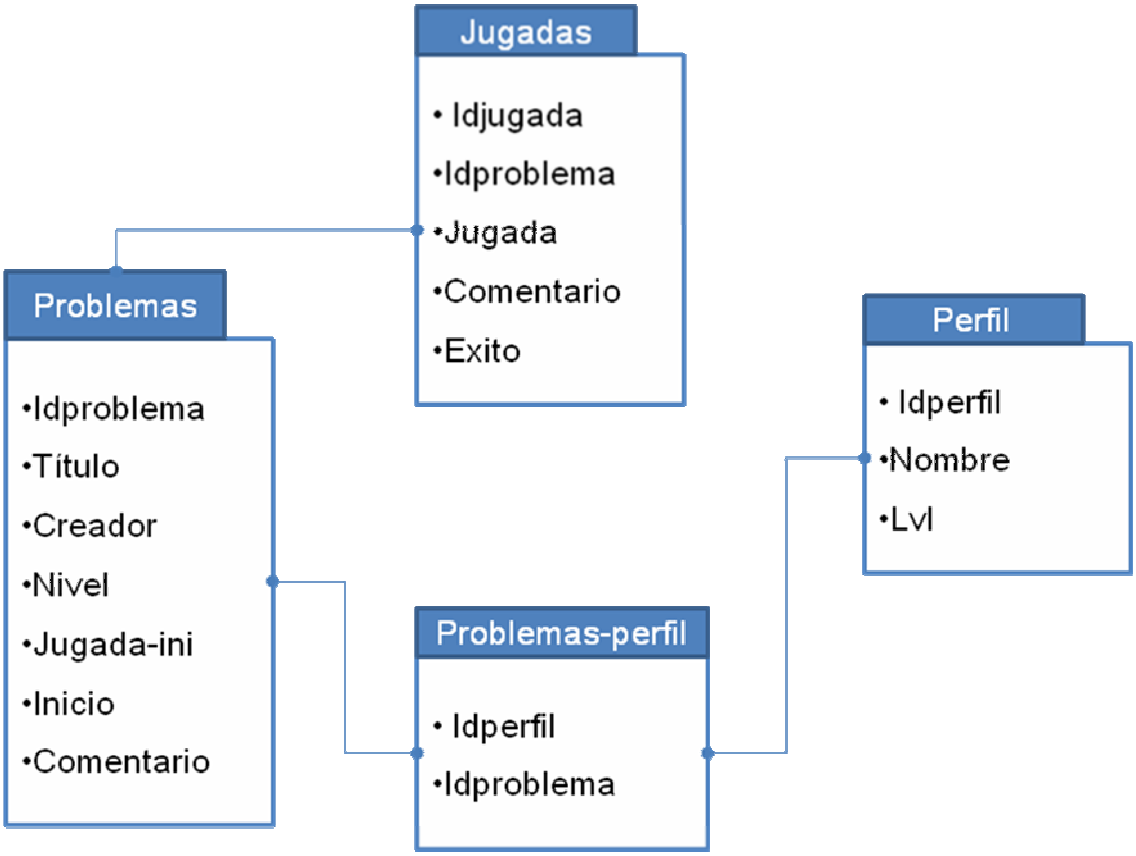
Problemas	
Clave primaria	Idjugada
Clave extranjera	Idproblema

Esta tabla almacena toda la información necesaria para definir un problema, para ello necesitamos los siguientes elementos:

- Idproblema: Identifica unívocamente al registro del problema en cuestión.
- Titulo: Indica el nombre del problema.
- Creador: Indica el creador del problema.
- Nivel: Indica el nivel mínimo que debe tener un perfil de usuario para poder jugar el problema.
- Jugada-ini: Contiene la situación inicial del problema.
- Inicio: Indica el jugador que será el usuario en la resolución del problema, blanco o negro.
- Comentarios: Contiene el comentario inicial del problema.

Elementos	*Atributos*
Idproblema	Es un entero de 11, clave primaria. Es auto incremental. No se permiten nulos.
Titulo	Es un varchar de 100 caracteres. Se permiten nulos.
Creador	Es un varchar de 50 caracteres. Se permiten nulos.
Nivel	Es un entero de 11. No se permiten nulos.
Jugada-ini	Es un varchar de 5000 caracteres. No se permiten nulos.
Inicio	Es un entero de 1. No se permiten nulos.
Comentario	Es un varchar de 2000 caracteres. Se permiten nulos.

Diagrama de la base de datos



En este diagrama se muestran las relaciones existentes entre las distintas tablas pertenecientes a la base de datos.

4. Programación de prototipos

Para la programación como se ha comentado, se va a usar Java en concreto la j2sdk-1_4_2_14 para Windows XP.

Java es un lenguaje de programación orientado a objetos. Las aplicaciones Java están típicamente compiladas en un bytecode, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el bytecode es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del bytecode por un procesador Java también es posible

El lenguaje Java se creó con cinco objetivos principales:

1. Usar la metodología de la programación orientada a objetos.
2. Permitir la ejecución de un mismo programa en múltiples sistemas operativos.
3. Incluir por defecto soporte para trabajo en red.
4. Ejecutar código en sistemas remotos de forma segura.
5. Ser fácil de usar y tomar lo mejor de otros lenguajes orientados a objeto

Java no implementa Bases de Datos, ya que solo es un lenguaje de programación, pero implementa funciones que permiten al programador realizar conexiones entre la interfaz de usuario y el Gestor de Base de Datos. La conexión entre la aplicación en java y la base de datos se realiza mediante los drivers, en este caso se va a usar como base de datos MySql por lo que el driver que se va usar es el mysql-connector-java-5.0.5.

MySQL es un sistema de gestión de base de datos relacional, multihilo y multiusuario con más de seis millones de instalaciones.

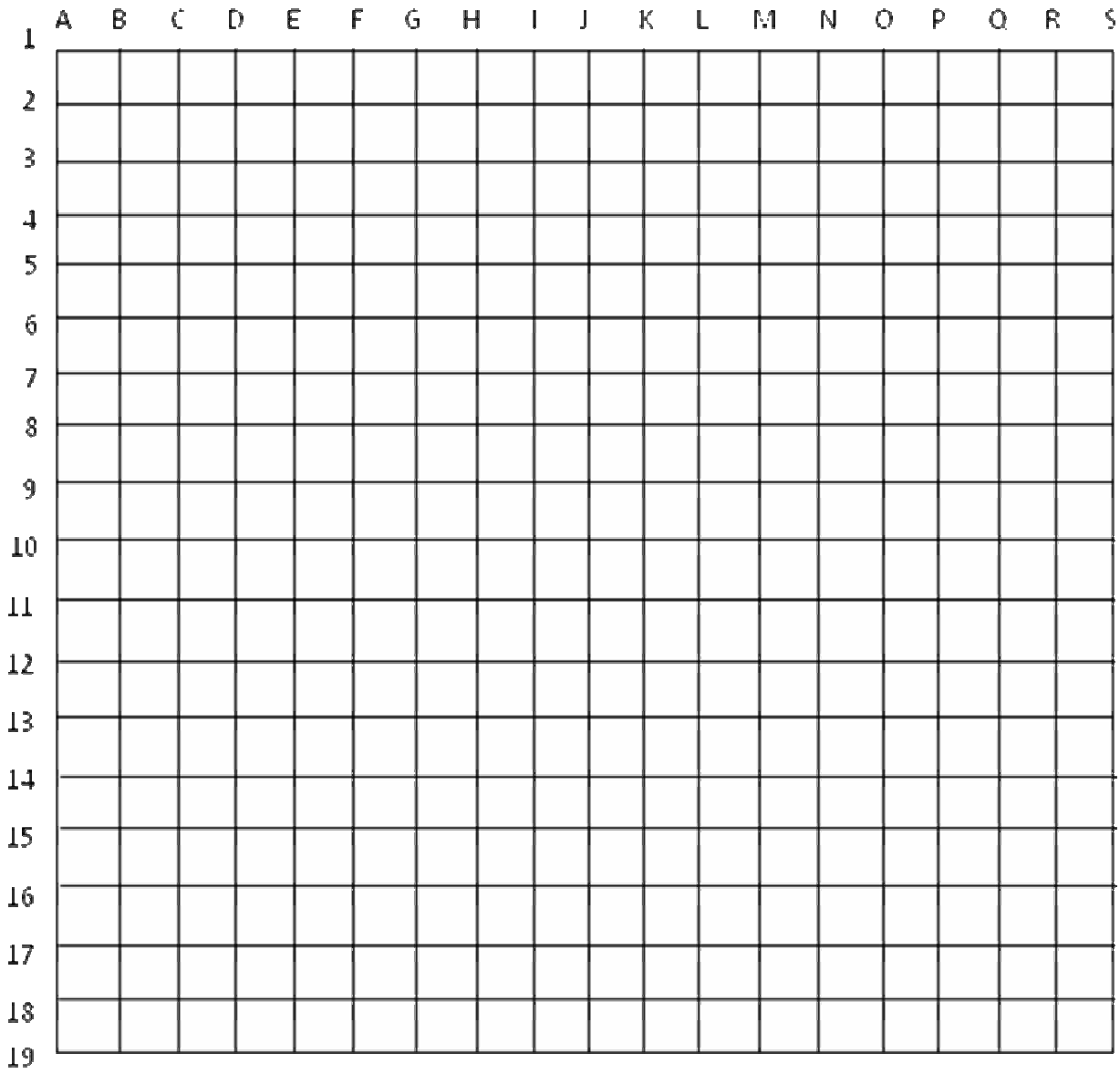
El entorno de desarrollo que se ha utilizado ha sido Eclipse

Una vez definidas las tecnología que se van a usar se definirán los distintos prototipos de la aplicación que se han realizado a lo largo del desarrollo del proyecto.

Para ello vamos a definir los objetivos que se han propuesto cumplir en este prototipo, como se han realizado los objetivos y por último evaluaremos el prototipo.

4.1. Prototipo I

En este prototipo de aplicación se va a desarrollar el tablero de juego. Se va a seguir el diseño realizado en la etapa anterior por lo que el tablero queda con el siguiente aspecto:



Para ello se han utilizado labels (etiquetas) con el siguiente fondo:

Este fondo se utiliza en los labels que se encuentran en el interior del tablero.

Este fondo se utiliza en la esquina superior izquierda del tablero.

Este fondo se utiliza en la esquina superior derecha del tablero.

Este fondo se utiliza en la esquina inferior izquierda del tablero.

Este fondo se utiliza en la esquina inferior derecha del tablero.

Este fondo se utiliza en el margen izquierdo del tablero.

Este fondo se utiliza en el margen derecho del tablero.

Este fondo se utiliza en el margen superior del tablero.

Este fondo se utiliza en el margen superior del tablero.

A continuación se muestra un ejemplo de cómo queda el label cuando se pincha sobre él.

El fondo del label en el caso de ser un label del interior del tablero cambia a este fondo en el caso de que le toque mover al jugador negro.

El fondo del label en el caso de ser un label del interior del tablero cambia a este fondo en el caso de que le toque mover al jugador blanco.

Estos cambios de label se repiten en todos los fondos de labels existentes.

4.2. Prototipo II [FROU07]

En este prototipo de aplicación se va a desarrollar el tablero de juego. Se va a seguir el diseño realizado en la etapa anterior, pero utilizaremos un técnica para el desarrollo del interfaz denominada técnica de doble-buffering.

Doble-buffering:

Al mostrar gráficos con las técnicas estándar, las imágenes suelen aparecer a trozos o con parpadeo. Las aplicaciones Java permiten que los programas dibujen en memoria, para luego ir mostrando la imagen completa de forma suave.

Este es el proceso conocido como *doble-buffering*, y tiene dos ventajas fundamentales sobre el proceso normal de pintar o dibujar directamente sobre la pantalla:

Primero, el usuario ve aparecer de golpe la imagen en la pantalla. Mientras el usuario está viendo esa imagen, el programa está generando la siguiente para mostrarla de golpe a continuación, y así una y otra vez.

Segundo, la técnica de doble-buffering involucra un objeto Image, que se puede pasar directamente a varios métodos. Esta capacidad para manipular objetos Image permite descomponer las rutinas de dibujo en componentes funcionales, en lugar de un enorme método *paint()*.

Para entender el doble-buffering, primero se necesita comprender qué es un contexto gráfico. Un contexto gráfico es simplemente una estructura de datos que el sistema sabe utilizar como tablero de dibujo, es decir, es la zona en que se va a pintar. Ya hemos visto y utilizado contextos gráfico en las declaraciones del método *paint()*:

```
public void paint( Graphics g ) {
```

El objeto Graphics g es el contexto gráfico. Se utiliza g para realizar todo el dibujo. Por ejemplo:

```
g.drawString( "¡Hola!",25,25 );
g.drawRect( 15,15,50,10 );
```

Entonces, Java traduce todo lo que se dibuja en g en imágenes sobre la pantalla. Para realizar doble-buffering, se necesita pues, primero crear un contexto gráfico que no es presentado inmediatamente en la pantalla.

Creación de contextos gráficos:

Crear contextos gráficos tiene dos pasos: Crear una imagen vacía con las dimensiones adecuadas y obtener un objeto Graphics de esa imagen. El objeto Graphics que se construye en el segundo paso realiza la función de contexto gráfico.

Por ejemplo:

```
import java.awt.*;
import java.applet.Applet;

public class CGrafico extends Applet {
    Image dobleBuffer;
    Graphics miCG;

    public void init() {
        // Inicializa el doble buffer
        dobleBuffer = createImage( 300,300 );
        miCG = dobleBuffer.getGraphics();

        // Construye un área gráfica de trabajo
        miCG .setColor( Color.white );
        miCG.fillRect( 0,0,300,300 );
        resize( 500,450 );
        }
```

Podemos utilizar `miCG` para dibujar cualquier cosa. Las imágenes se trazarán en doble buffer. Cuando el dibujo esté terminado, se puede presentar el doble buffer en pantalla:

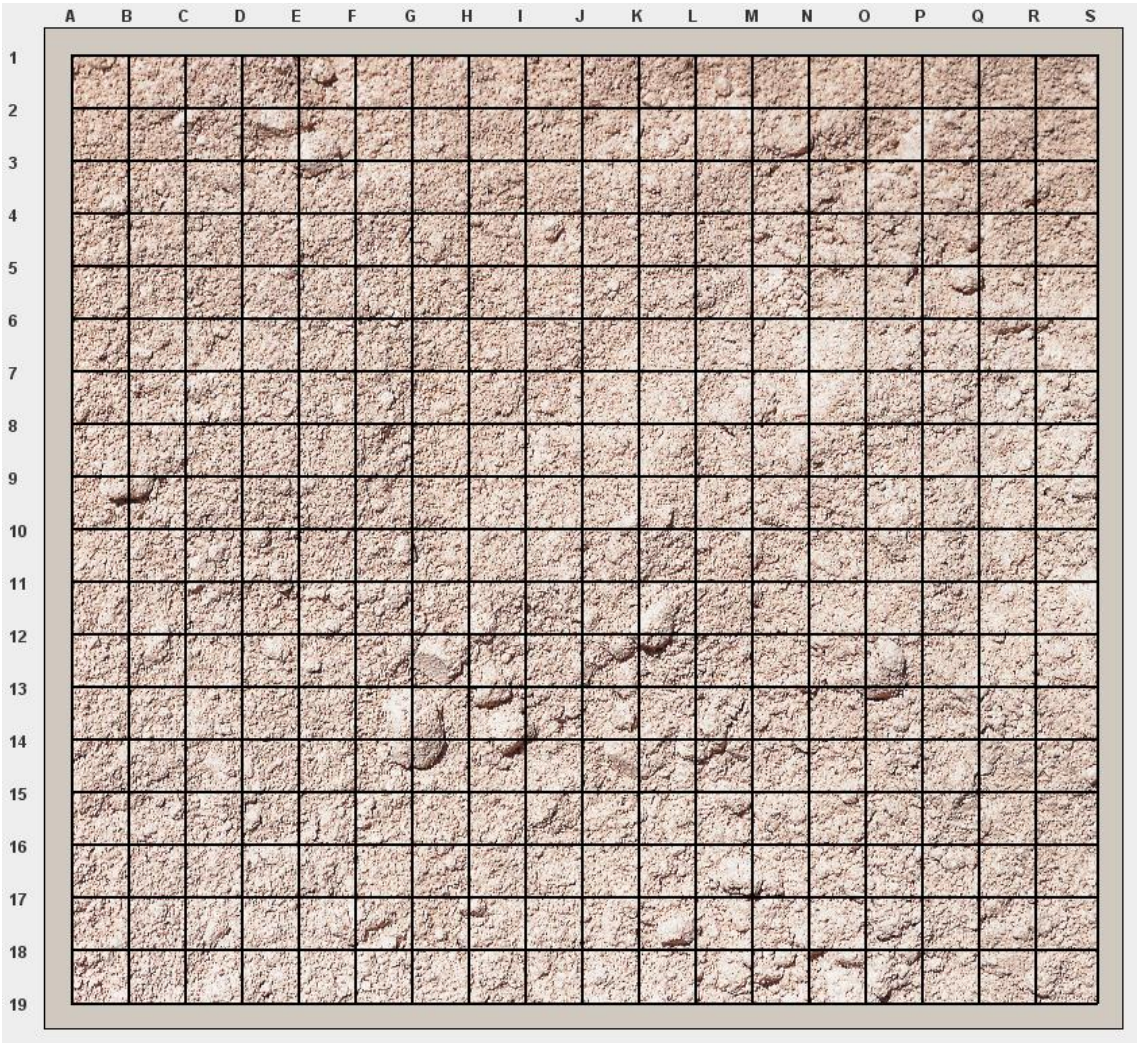
```
public void paint( Graphics g ) {
    // Sólo se tiene que presentar la imagen del buffer
    g.drawImage( dobleBuffer,0,0,this );
    }
```

Utilización de contextos gráficos:

Una vez definido un contexto gráfico, podemos usarlo en cualquier parte de nuestro programa. Por ejemplo, podemos repartir la responsabilidad para dibujar sobre varias funciones:

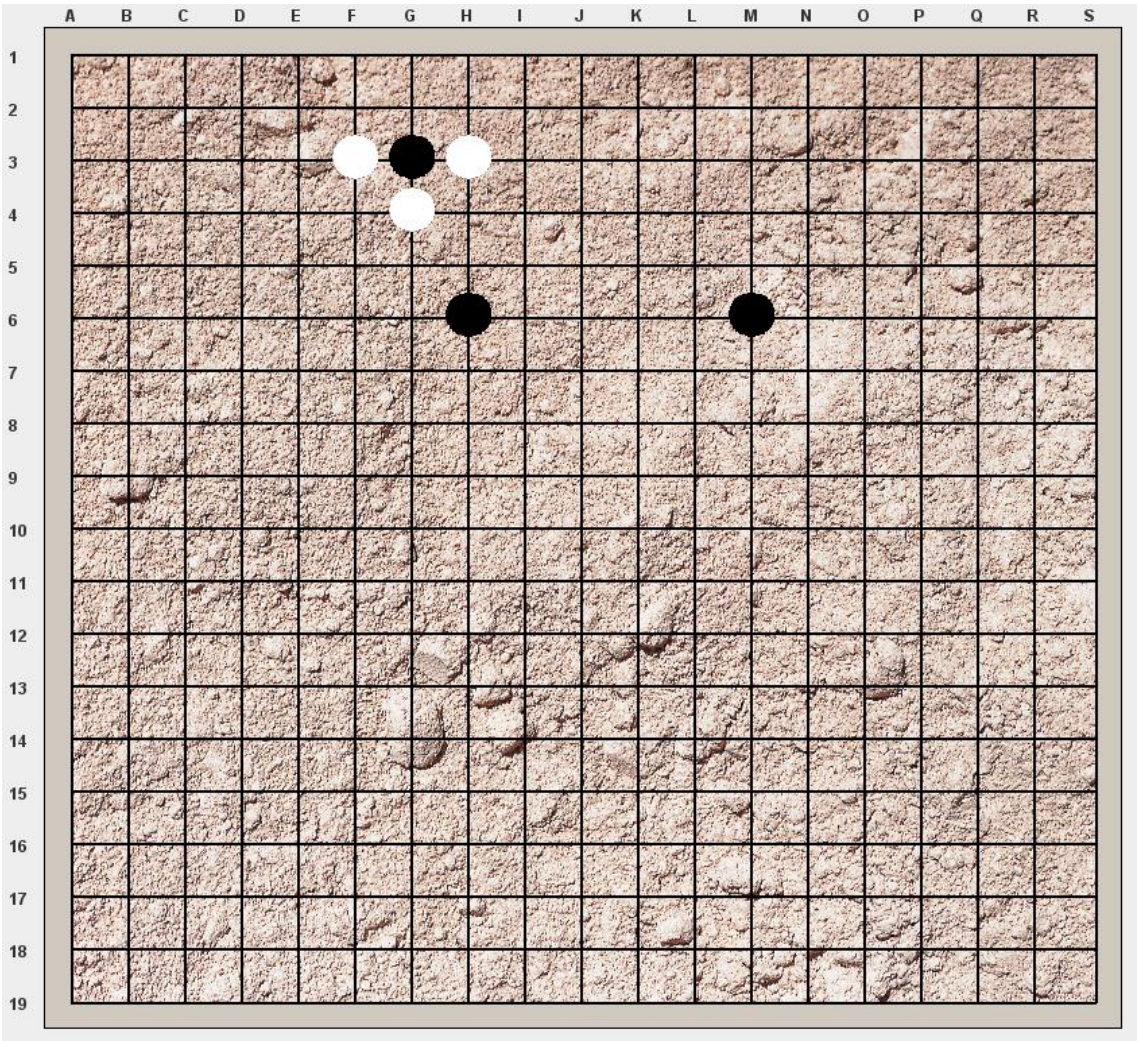
```
public void titulo() {
    // Obtiene la fuente de texto actual y la guardamos
    Font f = miCG.getFont();
    // Seleccionamos otra fuente para el título
    miCG.setFont( new Font( "TimesRoman".Font.BOLD,36 ) );
    miCG.drawString( "Ejemplo de Espiral",15,50 );
    miCG.drawString( "Círculos",15,90 );
    // Recuperamos la fuente original
    miCG.setFont( f );
   }
public void espiral() {
    int x,y;
    // Dibujamos circulos en los lados horizontales
    y = 100;
    for( x=100; x <= 200; x+=10 )
        {
        miCG.drawOval( x,y,20,20 );
        miCG.drawOval( x,y+100,20,20 );
        }
    // Ahora en los verticales
    x = 100;
    for( y=100; y <= 200; y+=10 )
        {
        miCG.drawOval( x,y,20,20 );
        miCG.drawOval( x+100,y,20,20 );
        }
    }
public void start() {
    // Hace el dibujo off-line
    titulo();
    espiral();
    // Ahora muestra la imagen de golpe
    repaint();}
```

Utilizando esta técnica se ha conseguido que el tablero quede con el siguiente aspecto:



La clase que genera el tablero que se observa en la imagen, hereda de la clase cambas, lo que nos permite poder dibujar en la imagen. Para ello debemos conocer las dimensiones del tablero ya que las líneas del tablero se pintan en base a una posición absoluta en el tablero.

A continuación mostraremos cómo se colocan las piezas en el tablero:

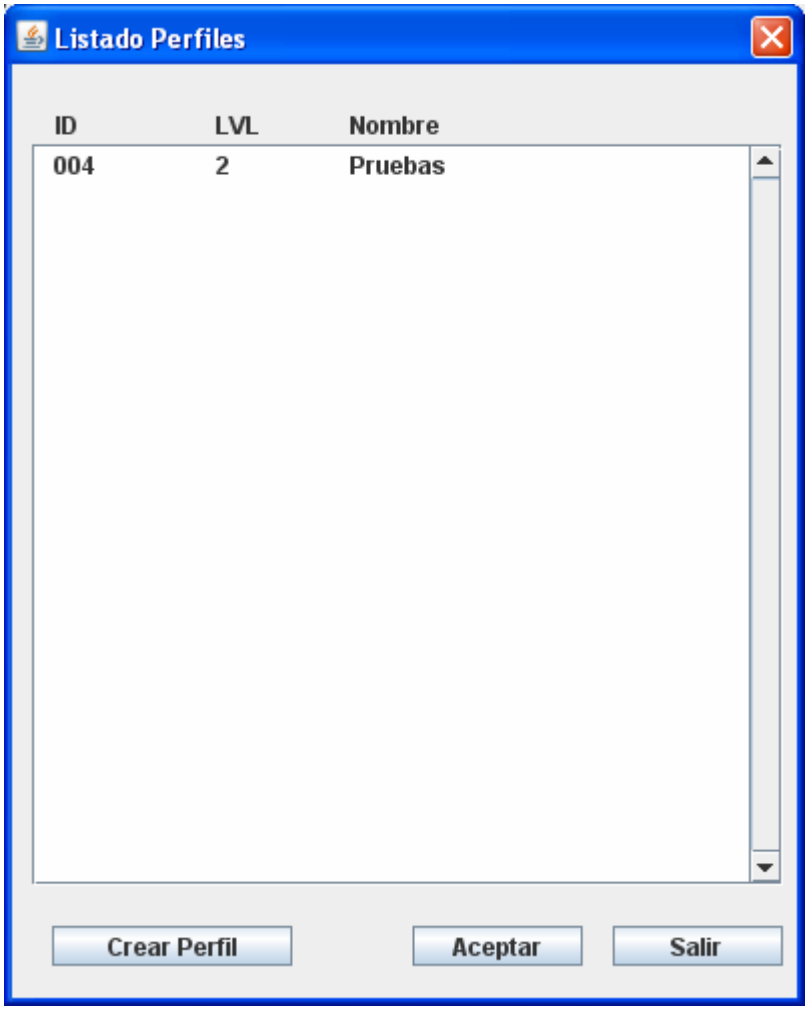


En esta imagen se observa como se colocan las piezas en las intersecciones, para ello se dispone de un método que traduce en el momento del click del ratón las coordenadas absolutas del ratón a unas coordenadas relativas que se utilizaran en la creación y representación de la piedra en el tablero

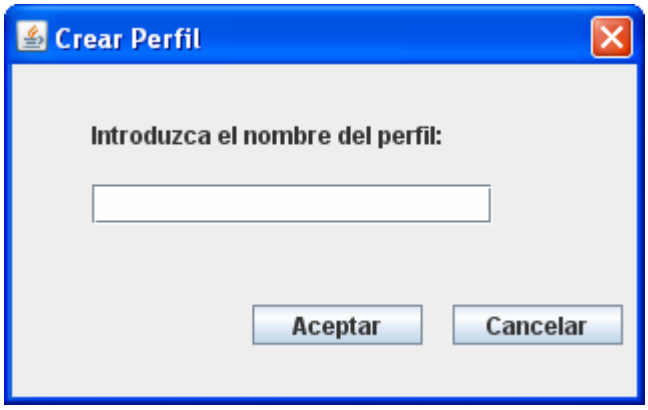
4.3. Prototipo III

En este prototipo evolucionará el objeto “tablero” que se realizo en el prototipo II integrándolo en una ventana basada en el diseño de la actividad anterior, para obtener así el entorno de juego deseado.
Además implementaremos las estrategias básicas de juego y la aplicación de problemas de juego.

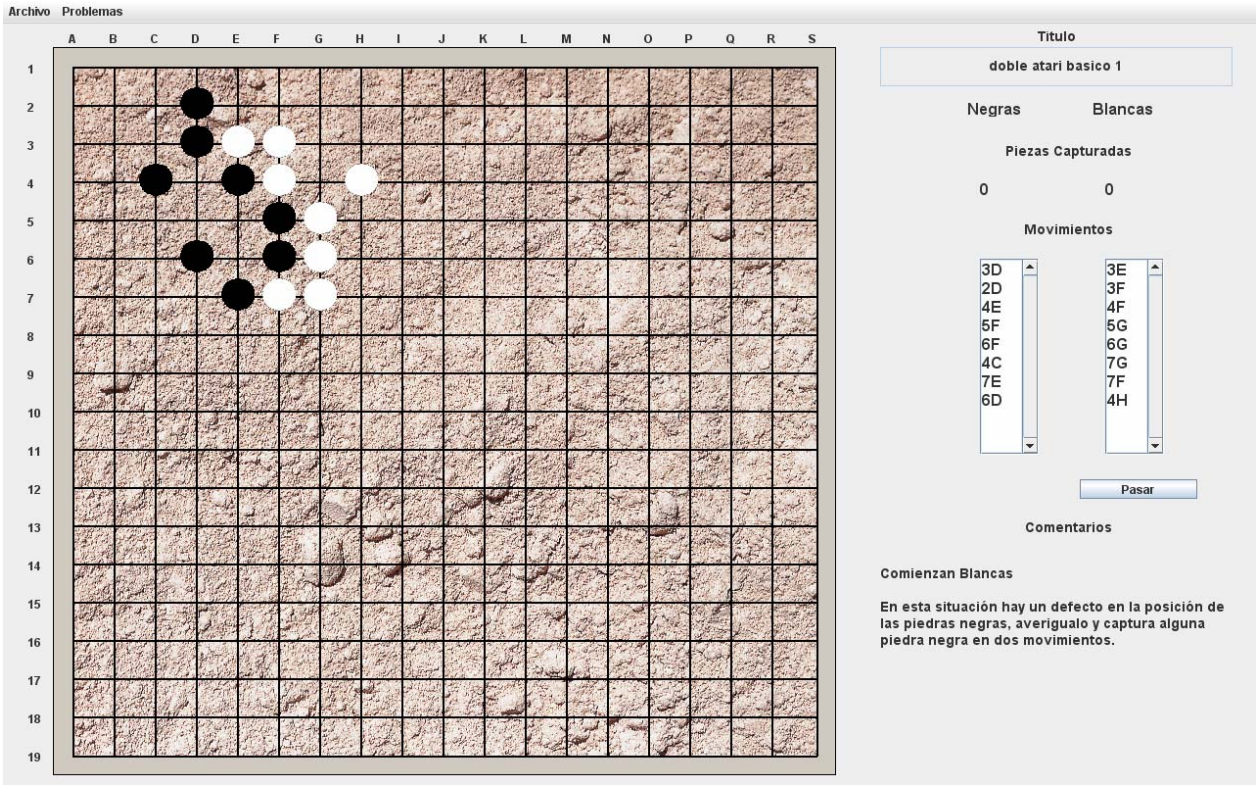
Antes de empezar a jugar necesitamos seleccionar o crear el perfil con el que vamos a jugar:



En este diálogo seleccionaremos el perfil con el que se va a jugar o crearemos el perfil con el que se quiera jugar.



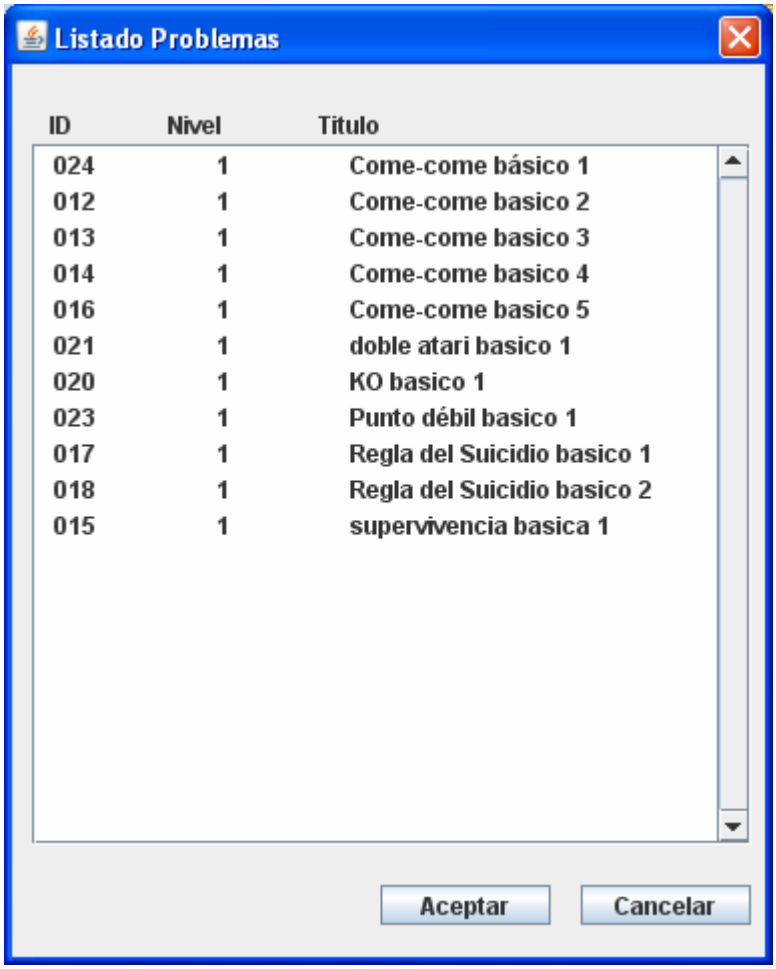
Con este diálogo crearemos el nuevo perfil con el que se quiere jugar.


Archivo Problemas
A B C D E F G H I J K L M N O P Q R S
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Titulo
doble atari basico 1
Negras
Blancas
Piezas Capturadas
0
0
Movimientos
3D 2D 4E 5F 6F 4C 7E 6D
3E 3F 4F 5G 6G 7G 7F 4H
Pasar
Comentarios
Comienzan Blancas
En esta situación hay un defecto en la posición de las piedras negras, averigualo y captura alguna piedra negra en dos movimientos.

Este es el entorno de juego, como podemos observar tenemos dos menús en la parte superior izquierda de la imagen, que nos permiten salir y seleccionar un problema de juego respectivamente.

En la parte derecha de la imagen tenemos el título del problema seleccionado, como se encuentra la situación de la partida y los comentarios correspondientes del problema.

Para seleccionar un problema lo haremos a través del diálogo que mostramos a continuación


Listado Problemas
ID
Nivel
Titulo
024 1 Come-come básico 1
012 1 Come-come basico 2
013 1 Come-come basico 3
014 1 Come-come basico 4
016 1 Come-come basico 5
021 1 doble atari basico 1
020 1 KO basico 1
023 1 Punto débil basico 1
017 1 Regla del Suicidio basico 1
018 1 Regla del Suicidio basico 2
015 1 supervivencia basica 1
Aceptar
Cancelar

Una vez seleccionado el problema, le damos al botón aceptar, a continuación la aplicación accede a la base de datos y carga en la aplicación el problema en cuestión.

Una vez cargado el problema, mostramos en los labels de título y comentarios la información correspondiente del problema. A continuación recorremos los movimientos de la jugada inicial del problema, se envía movimiento por movimiento al módulo de reglas que verifica que se cumplen las reglas del juego y los convertimos a las coordenadas relativas que puede representar el tablero.

A continuación explicaremos el funcionamiento de los objetos más importantes para la aplicación de la lógica en este prototipo.

Reglas: A este módulo le pasamos la cadena de piedras que componen las piedras representadas en el tablero y la piedra que se quiere colocar, no es un objeto en sí, sino que está integrado en el tablero. Recorre las piedras de las cadenas de piedra comprobando que si se cumple alguna de las siguientes reglas:

1) Cuando un jugador hace una jugada que priva de su última libertad a una piedra o formación del oponente debe sacar las piedras rodeadas del tablero y guardarlas separadamente hasta el final de la partida.
2) No está permitido hacer una jugada ocupando la última libertad en el interior de una formación enemiga (suicidio) a no ser que esta jugada capture una o más piedras enemigas.
3) Si un jugador captura una piedra en situación de ko, el otro jugador no puede recapturar inmediatamente. Ha de hacer otra jugada antes de recapturar.

Consecuencias:

- En el caso de que se dé la opción 1) se llamará al método actualizarCadenasyAlrededores() en el que eliminaremos las cadenas que no tengan libertades y una vez eliminadas éstas se actualizan las libertades de las cadenas de los alrededores.
- En el caso de que se dé la opción 2) no se permitirá el movimiento.
- En el caso de que se dé la opción 3) no se permitirá el movimiento.

Pieza: Es el objeto básico de juego, contiene toda la información necesaria para definir una pieza, es decir, el color de la pieza, su ubicación y las libertades que posee. También contiene los siguientes métodos para interactuar con ella:

- public int getX(): devuelve la posición relativa al eje de abscisas.
- public int getY(): devuelve la posición relativa al eje de ordenadas.
- public boolean getColor(): devuelve el color de la pieza en valores booleanos, true para negro y false para blanco.

Cadena: Es un conjunto de piezas conectadas. Una está conectada a otra cuando son del mismo color y las intersecciones en las que están posicionadas están conectadas por una línea. Por lo que una cadena solo puede ser de un color. Para definir una cadena se necesita una colección de piezas conectadas y que sean todas del mismo color y las libertades de la cadena. Los métodos más importantes que tenemos para interactuar con esta clase son los siguientes:

- public void añadirPieza(Pieza param): Permite añadir una pieza, para ello se vuelve a calcular las libertades de la cadena una vez añadida la pieza.
- private boolean comprobarPertenenciaPieza(Point punto): Comprueba si existe la pieza en la cadena.
- private boolean comprobarPertenenciaCadena(Point punto): Comprueba la pertenencia de la pieza a la cadena.
- public int getLibertades(): devuelve las libertades de la cadena.
- public boolean getColor(): Devuelve el color de la cadena.
- public ArrayList getPiezas(): Devuelve la colección de piezas que pertenecen a la cadena.
- public void borrarPieza(Pieza p): Elimina de la colección de piezas de la cadena la pieza en cuestión.
- public void combinarCadenas(ArrayList param,Cadena param1,int ordenada,int abcisa,ArrayList lista): Este método me permite combinar distintas cadenas del mismo color.

-

- public void actualizarCadena(): Este método me permite actualizar las libertades de la cadena.
- public int numPertenencia(Pieza p): Este método me permite calcular el número de pertenencia de una pieza a una cadena, esto es útil para calcular el número de libertades de la cadena y en consecuencia de la pieza.

AccesoSql: Este objeto contiene todas las instrucciones necesarias para interactuar con las tablas de problemas, perfiles de usuario y jugadas tanto para lectura, creación, borrado o modificación de los registros que contiene. Los métodos más importantes que tenemos para interactuar con esta clase son los siguientes:

- public void conectar(): Este método realiza la conexión con la base de datos.
- public ResultSet listarProblemas(): Este método nos devuelve un resultset con el listado de problemas.
- public ResultSet listarPerfiles(): Este método nos devuelve un resultset con el listado de perfiles.
- public Problema obtenerProblema(int id): A este método se le pasa la id del problema que queremos conseguir y nos lo devuelve.
- public Problema obtenerPerfil(int id): A este método se le pasa la id del perfil que queremos conseguir y nos lo devuelve.
- public void insertarProblemaJugado(int param1,int param2): Una vez que se ha completado un problema con éxito llamamos a este método para reflejarlo en la base de datos.
- public boolean actualizarLvlPerfil(int id): Mediante este método podemos actualizar el level de un perfil, se subirá de nivel cuando el mismo perfil resuelva con éxito 15 problemas.

Jugada: Este objeto contiene todo lo necesario para la definición de una jugada perteneciente a un problema, para ello almacena la información de la jugada que se va a jugar. Esta información se compone de:

- Identificador de la jugada.
- Identificador del problema.
- Secuencia de movimientos que representa la jugada.
- Comentario de la jugada.
- Un indicador de si la jugada es la correcta o no.

Los métodos más importantes que tenemos para interactuar con esta clase son los siguientes:

- public int getId(): Devuelve el identificador de la jugada.
- public int getIdproblema(): Devuelve el identificador del problema al que pertenece la jugada.
- public String getJugada: Devuelve la secuencia de movimientos de la jugada.
- public String getComentarios(): Devuelve los comentarios de la jugada.
- public int getExito(): Devuelve un indicador numérico para saber si la jugada resuelve el problema. Devuelve un 1 si es una jugada correcta y un 0 en caso contrario.

Problema: Este objeto contiene todo lo necesario para la definición de un problema, para ello almacena la información del problema que se va a jugar. Esta información se compone de:

- Identificador del problema.
- Jugada inicial del problema.
- Comentario para el inicio del problema.
- Color del jugador al comenzar el problema.
- Una colección de las jugadas que componen el problema.

Los métodos más importantes que tenemos para interactuar con esta clase son los siguientes:

- public int getId(): Devuelve el id del problema.
- public String getTitulo(): Devuelve el título del problema.
- public String getCreador(): Devuelve el creador del problema.
- public String getJugadaIni(): Devuelve la jugada inicial del problema
- public int getInicio(): Devuelve un valor numérico que representa el color del usuario.
- public ArrayList getJugadas(): Devuelve la colección de jugadas que pertenecen al problema.

AplicarProblema: Este objeto contiene todo lo necesario para aplicar un problema, la estructura de este objeto se divide en tres partes. La primera establece el escenario inicial del juego, la segunda comprueba que existe una jugada en la que se está cumpliendo con la secuencia de movimientos, la tercera indica el siguiente movimiento a realizar. En la tercera parte continuamos la jugada en caso de encontrar alguna concordancia u obtenemos el siguiente movimiento que vamos a aplicar del módulo de estrategia. Los métodos más importantes que tenemos para interactuar con esta clase son los siguientes:

- public void iniciar(Problema p): Este método recorre los movimientos de la jugada inicial del problema, los convierte al formato para que lo entienda el objeto "tablero" y se lo da para que lo represente.
- public void comprobar(int param1,int param2): Este método comprueba si el movimiento que ejecuta el usuario coincide con el movimiento de alguna jugada.
- public void sigMovimiento():Este método comprueba si la maquina puede ejecutar el siguiente movimiento de la jugada o tiene que llamar al módulo de estrategia para ejecutar el siguiente movimiento.
- public boolean ultimo(): Este método devuelve un valor booleano según sea el último movimiento o no. Devuelve "true" si es el último movimiento y "false" en caso contrario.

Estrategia: Este objeto contiene los algoritmos necesarios para aplicar las estrategias básicas del juego. Los métodos más importantes que tenemos para interactuar con esta clase son los siguientes:

- public void movimiento(): Ejecuta una de las 21 reglas de producción posibles, estas reglas tienen en cuenta en primer lugar la situación de las piezas con las que juega la máquina y en segundo lugar la posición de las piezas del contrario.

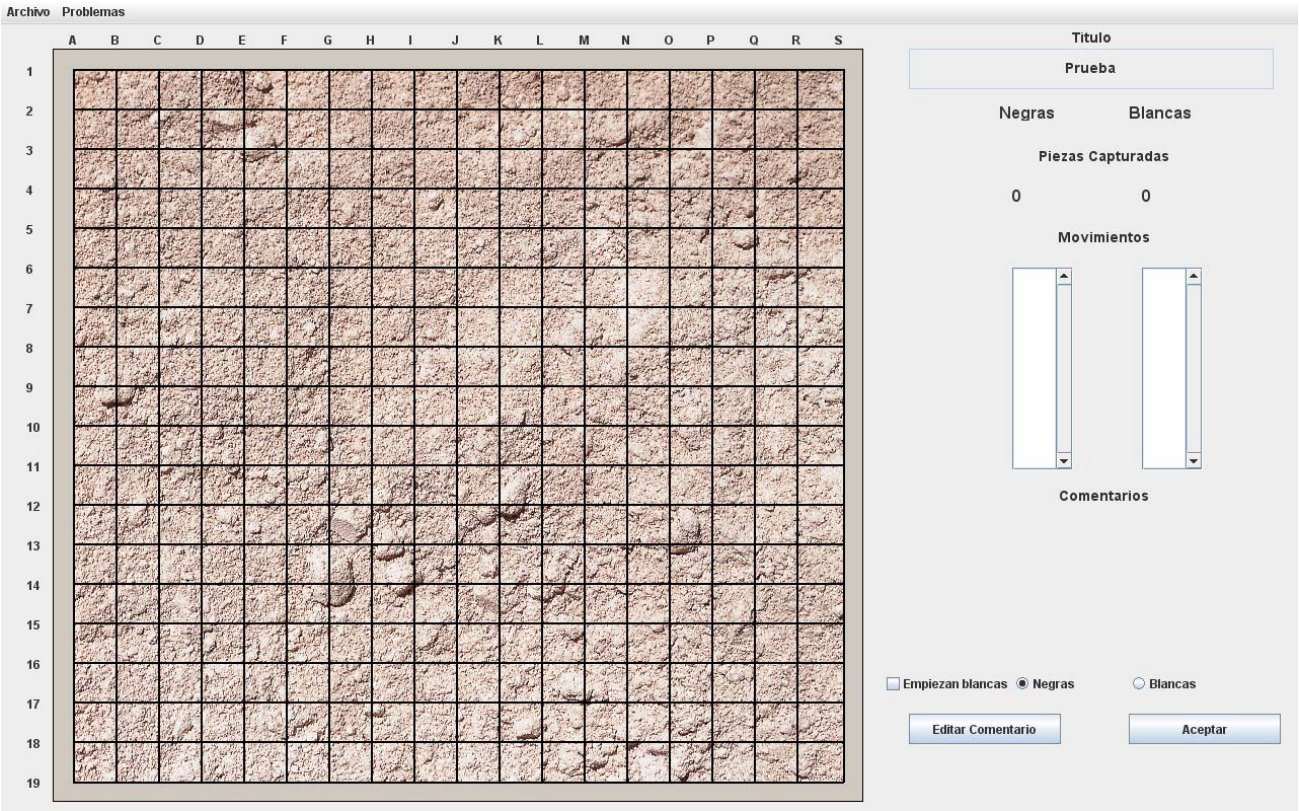
  Como primer objetivo tiene su supervivencia, y como segundo objetivo tiene el capturar las piezas rivales.
- private void comprobarExistencia(Point p): Comprueba que no exista una pieza en el tablero en ese punto.
- public void posible(Point p): Estudiando el posicionamiento del tablero comprueba que es posible colocar una pieza en el punto indicado.
- public void buscarCadenaCercana(Point p): Busca todas las cadenas cercanas al punto en el que se quiere colar la pieza.
- public void buscarCadenaOtroColor(Point p): Busca las cadenas cercanas del adversario al punto indicado.
- public void supervivencia(Cadena cad): Este método busca la mejor forma de colocar una pieza para la supervivencia de la cadena indicada.
- public void comer(Cadena cad): Este método busca un punto viable en el que colocar una pieza y capturar una cadena del adversario, o en su defecto, para reducir el número de libertades de la cadena del adversario.

4.4. Prototipo IV

En este prototipo implementará el objeto “tablero” que se realizó en el prototipo III integrándolo en una ventana basada en el diseño de la actividad anterior, para obtener así el editor de problemas deseado.

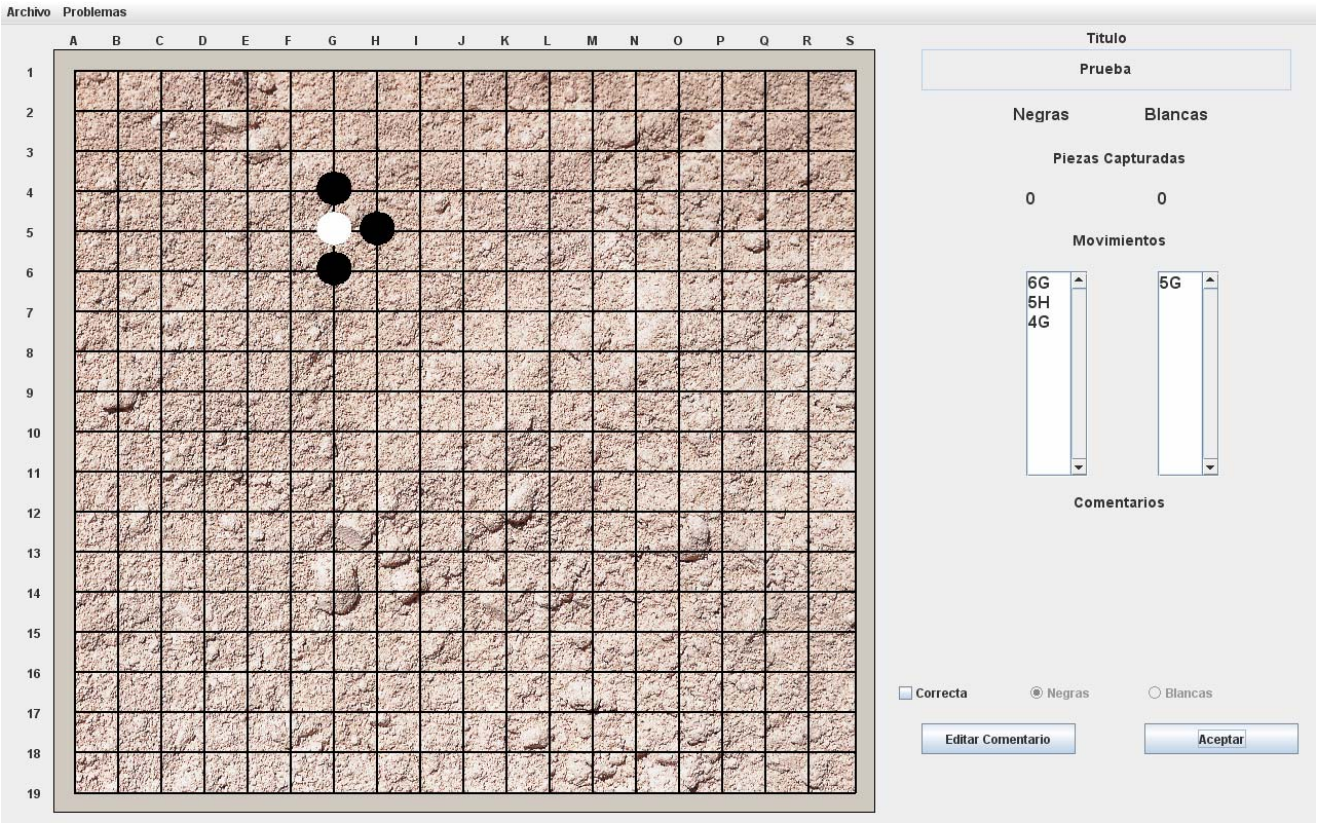
Además implementaremos los objetos necesarios para la edición de problemas.

A continuación mostraremos la ventana principal en el editor de problemas:



Esta ventana difiere en ventana de juegos en que en ésta tenemos los botones de aceptar y de editar comentar, el checkbox “Empiezan blancas” y los radiobuttons “Negras” y “Blancas”.

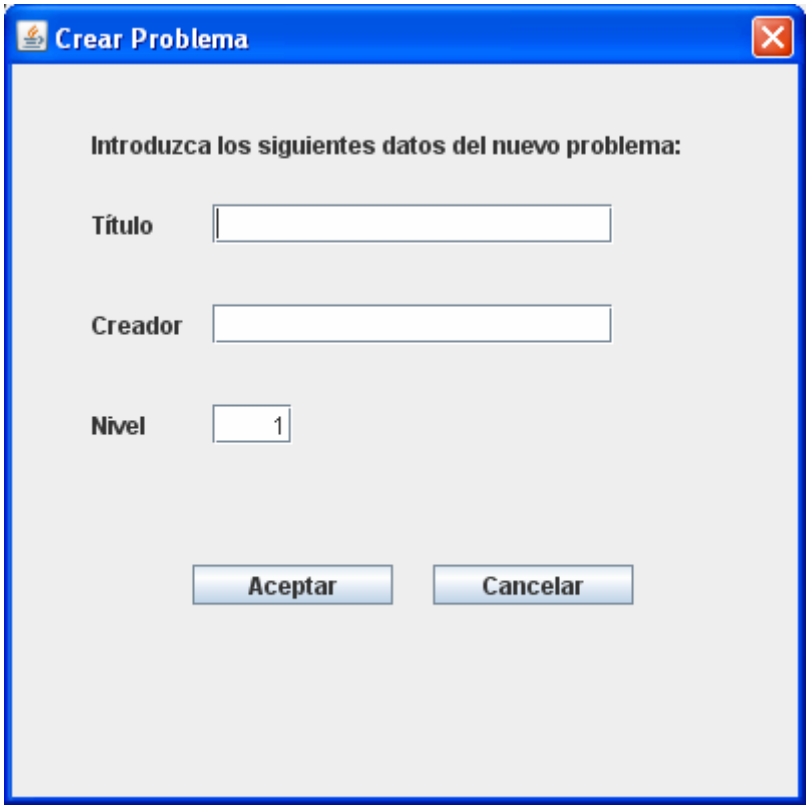
Comenzaremos introduciendo jugada inicial del problema en el tablero y añadiendo un comentario para que se muestre al comienzo del problema, a continuación se pulsa el botón “Aceptar” y guardamos en la aplicación el problema con la jugada inicial.



Como vemos después de introducir la jugada inicial el checkbox ha cambiado, ahora permite indicar si la jugada que vamos a introducir es correcta o no.

Una vez desarrollada una jugada, le damos a aceptar y volveremos a la situación inicial del problema lo que nos permitirá desarrollar más jugadas y añadir los respectivos comentarios.

En el menú superior tenemos la opción problema y dentro de ésta la opción “Guardar” que permite guardar el problema junto con todas las jugadas creadas en la base de datos MySql.



Esta ventana es la inicial que se presenta cuando un usuario quiere crear un problema, en ella el usuario debe introducir el título del problema que va a crear, el nombre del creador del problema, y asignarle un nivel al problema, por defecto el nivel asignado a un problema es 1.

A continuación explicaremos el funcionamiento de los objetos más importantes para la aplicación de la lógica en este prototipo:

Reglas: Este objeto es similar al explicado en el prototipo anterior, con la salvedad de que cuando el usuario introduce la jugada inicial, puede elegir el color de la pieza a colocar.

AccesoSql: Este objeto contiene todas las instrucciones necesarias para interactuar con las tablas de problemas y jugadas tanto para lectura, creación, borrado o modificación de los registros que contiene. Los métodos más importantes que tenemos para interactuar con esta clase son los siguientes:

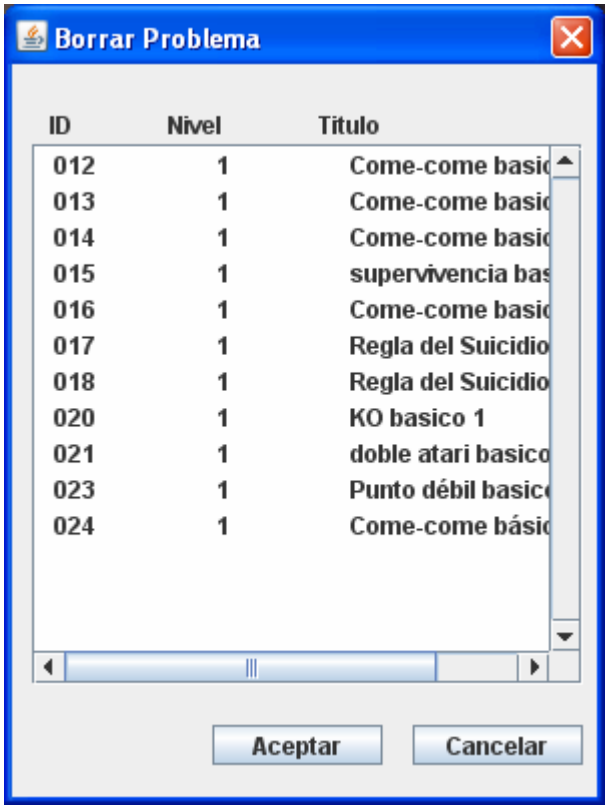
- public void insertarProblema(Problema p): Una vez creado un problema, este método nos permite añadirlo a la base de datos.

AplicarJugadaIni: Este objeto nos permite que cada vez que se pulse el botón "Aceptar" una vez introducida una jugada inicial aparezca representada en el tablero la jugada inicial.

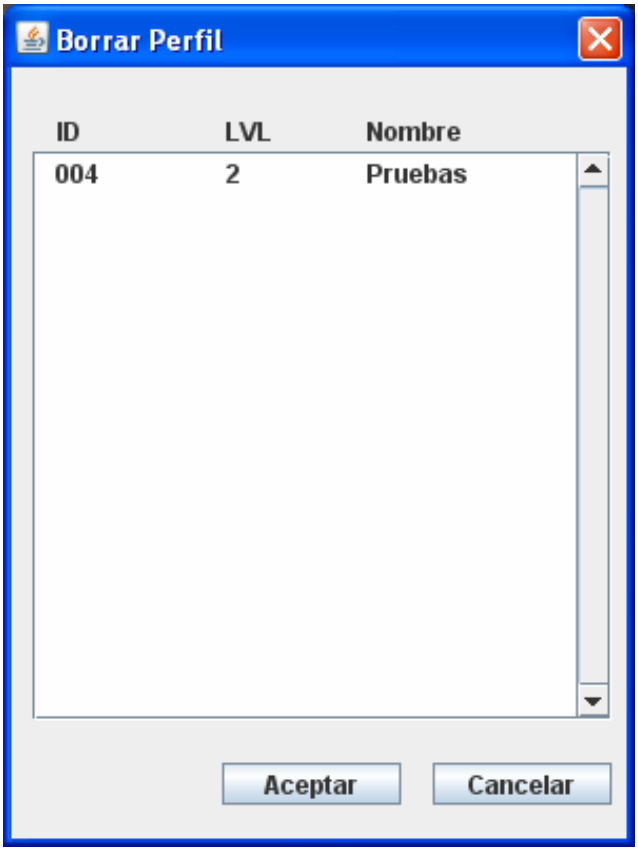
Prácticamente toda la lógica del editor de problema recae en la acción de pulsar el botón aceptar, ya que si los movimientos cumplen las reglas o no es algo que tiene integrado el tablero y éste se realizo en el prototipo III.

4.5 Prototipo V

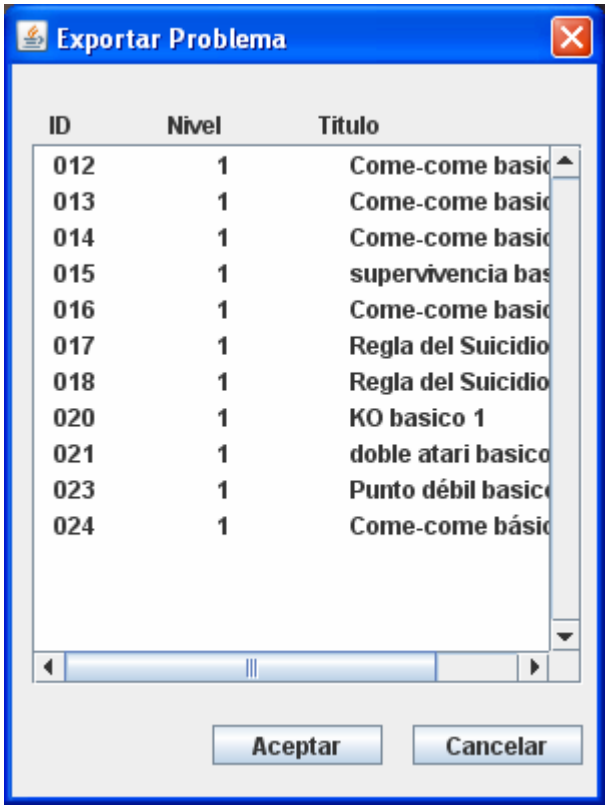
En este prototipo se han creado todos los diálogos necesarios para la gestión del juego y una ventana inicial para la integración del juego del Go, el editor de problemas y los diálogos para la gestión. A continuación mostraremos los diálogos junto con una breve descripción de su funcionamiento:



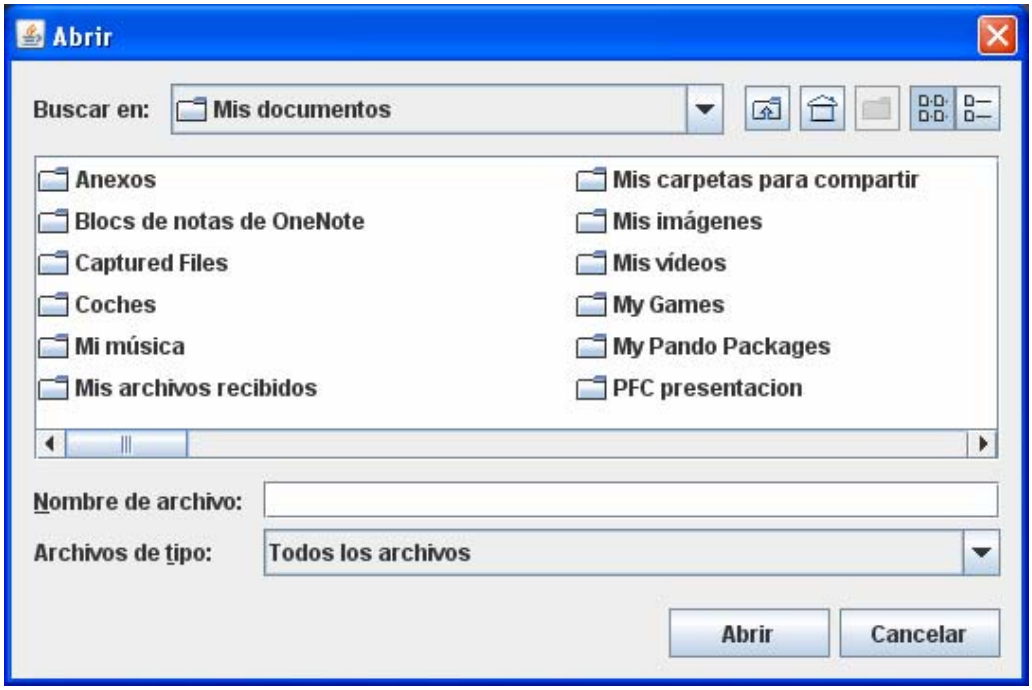
Este diálogo me permite, sólo con seleccionar un problema y pulsar “Aceptar”, eliminar un problema de la base de datos. Para ello realizamos una conexión con la base de datos a través del objeto AccesoSql y llamamos al método borrarProblema(int), pasándole el identificador del problema.



Este diálogo me permite, sólo con seleccionar un perfil y pulsar "Aceptar", eliminar un perfil de la base de datos. Para ello realizamos una conexión con la base de datos a través del objeto AccesoSql y llamamos al método borrarPerfil(int), pasándole el identificador del perfil.



Este diálogo me permite, sólo con seleccionar un perfil y pulsar "Aceptar", exportar un problema a un fichero externo que se almacena en el directorio que contiene la aplicación. Para ello realizamos necesitamos, que la clase problema y jugadas sea serializable, lo que simplifica significativamente el proceso de exportado e importado de problemas. La extensión que tiene el fichero es ".urb", lo que nos restringir los archivos que se pueden importar a archivos ".urb".



Este es el diálogo que nos permite importar problemas, con este diálogo seleccionamos el problema que queremos importar y devuelve un File con la url del fichero que se pretende importar.

Esta es la ventana inicial, en la que tenemos un menú con las diferentes funciones que ofrece esta aplicación.

5. Herramientas utilizadas

- Herramientas Hardware:
  - Portatil Dell Inspiron 6400
  - Procesador: Intel Centrino Duo a 1'66 Ghz
  - Memoria: 2 GB DDR2
  - Disco Duro: 60 GB (5400 rpm)

- Herramientas Software:
  - Sistema operativo: Windows XP
  - Se ha utilizado como lenguaje de programación Java y como entorno de desarrollo Eclipse.
  - Se ha utilizado para almacenar toda la información referente a los problemas y a los perfiles de usuario MySql.
  - Para el desarrollo de la documentación se ha utilizado Microsoft Word 2007.
  - Para el desarrollo del presupuesto se ha utilizado Microsoft Excel 2007.

6. Pruebas

Las pruebas constituyen una actividad importante en el desarrollo del proyecto, ya que gracias a ellas podremos descubrir los posibles errores cometidos durante el desarrollo del proyecto, y así garantizar la calidad del proyecto desarrollado.

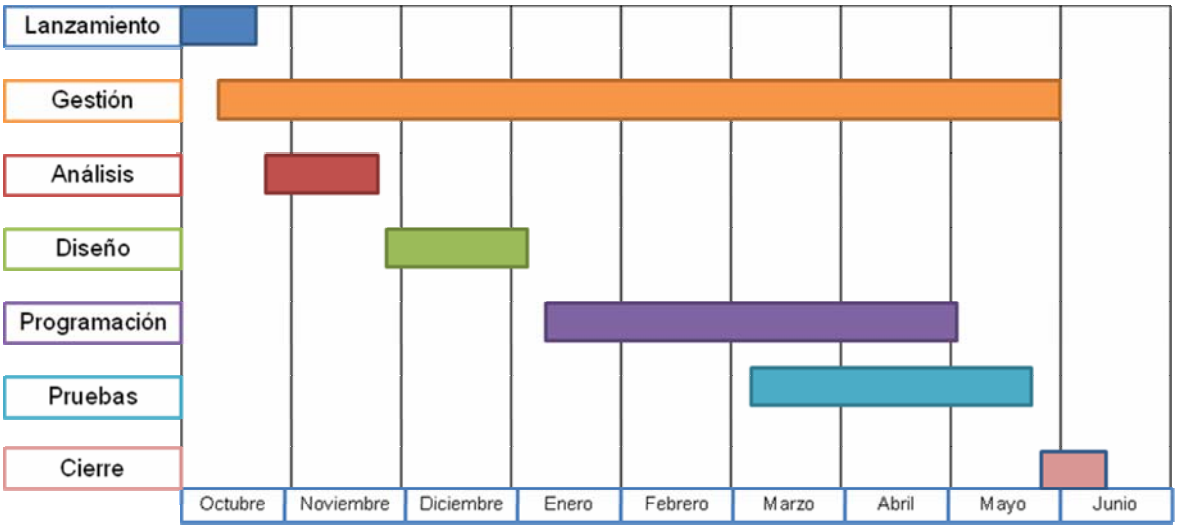
Las pruebas se han realizado durante todo el desarrollo de la parte de programación, probando por separado los distintos módulos que lo componen. Con estas pruebas se ha buscado encontrar una fiabilidad aceptable al término del proyecto y al poner todas las funcionalidades en conjunto. Estas pruebas son imprescindibles ya que facilitan la puesta en conjunto de los distintos módulos del proyecto. Además, al estar las pruebas acotadas los posibles errores son más fáciles de detectar.

Una vez finalizada la aplicación se ha procedido a la creación de problemas y a la resolución de los mismos. También se han probado el resto de funcionalidades por separado durante un periodo de una semana, para garantizar el mayor grado de fiabilidad posible.

7. Planificación

Para la planificación se ha utilizado un diagrama de Gantt en el que se puede ver la duración de las distintas actividades que componen el proyecto.

En el eje vertical están representadas las actividades, y en el eje horizontal está representada la escala de tiempos.



Estas actividades se han descrito a lo largo del presente documento

8. Presupuesto

Actividades	Horas	Coste
Lanzamiento	6	240
Gestión	15	600
Identificación de necesidades	10	350
Análisis de requisitos	15	525
Diseño	60	2100
Programación	230	6900
Pruebas	25	750
Cierre	5	200
Dirección del proyecto	25	1125
Coordinación del proyecto	10	500
Total	401	13290

Perfiles del proyecto

- Jefe de proyecto: Antonio Francisco Urbano Muñoz. Coste: 40 euros/hora.
    - Actividades que realiza:
        - Lanzamiento.
        - Gestión
        - Cierre
- Analista: Antonio Francisco Urbano Muñoz. Coste: 35 euros/hora.
    - Actividades que realiza:
        - Identificación de necesidades.
        - Análisis de requisitos.
        - Diseño
- Programador: Antonio Francisco Urbano Muñoz. Coste: 30 euros/hora
    - Actividades que realiza:
        - Programación.
        - Pruebas.
- Director del proyecto: José María Rodríguez Fernández. Coste: 45 euros/hora. Realiza la actividad de dirección de proyecto.
- Coordinador de proyecto: Miguel Ángel Sanz Bobi. Coste: 50 euros/hora. Realiza la actividad de coordinación del proyecto.

9. Conclusiones

Este proyecto causará un impacto en los juegos de estrategia, introduciendo en este mercado el primer programa inteligente de aprendizaje para el juego del Go.

La realización del proyecto me ha permitido aplicar parte mi conocimiento adquirido durante la carrera de ingeniería informática.

Futuras mejoras:

1.2. Permitir a los usuarios configurar el apartado gráfico.

1.3. Introducir estrategias de juego más complejas para la CPU.

1.4. Aplicar distintas estrategias de juego de la CPU según el nivel del usuario.

1.5. Soportar distintas resoluciones de pantalla.

10. Bibliografía

[BARR01] Jesús Barranco de Areba, *"Metodología del análisis estructurado de sistemas"*. Editorial Universidad Pontificia de Comillas

[WIKI07] http://es.wikipedia.org La enciclopedia libre.
La información de esta web se ha realizado por una comunidad de usuarios.

[MARQ01] María Mercedes Marqués Andrés *"Modelo entidad-relación"*
http://www3.uji.es/~mmarques/f47/apun/node83.html

[FROU07] Agustín Froufe *"Tutorial de Java"*
http://www.publispain.com/supertutoriales/diseno/java/cursos/3/Intro/tabla.html

[ANRI04] Jose Antonio Andújar Jovell y Juan Carlos del Rio, *"El juego del Go"*. http://www.lsi.upc.es/~jandujar/go

Anexo A

-Manual de Instalación-

Manual de instalación

Para la instalación de la aplicación hay que ejecutar el archivo install.bat que se encuentra en la carpeta “Aplicación/Go” del cd de instalación. De esta manera se ejecutan de forma secuencial los distintos archivos ejecutables necesarios para que la aplicación Go funcione correctamente.

Los archivos ejecutables se encuentran en la carpeta “Aplicación/Go/Aplicaciones” del cd de instalación y son los siguientes:

- j2sdk-1_4_2_14-windows-i586-p-iftw
- mysql-essential-5.2.3-falcon-alpha-win32
- setup
- Sql

En la instalación de los archivos necesarios seguiremos los pasos que nos indican los instalables, salvo en el siguiente paso:

- Al instalar base de datos MySql se elegirá la configuración estándar y ejecutaremos “configurar ahora”, el password que daremos al perfil root es el de: celtas.

Por último solo queda ejecutar el link que se encuentra en el escritorio, llamado Go.

Anexo B

-Manual de Usuario-

Manual de usuario

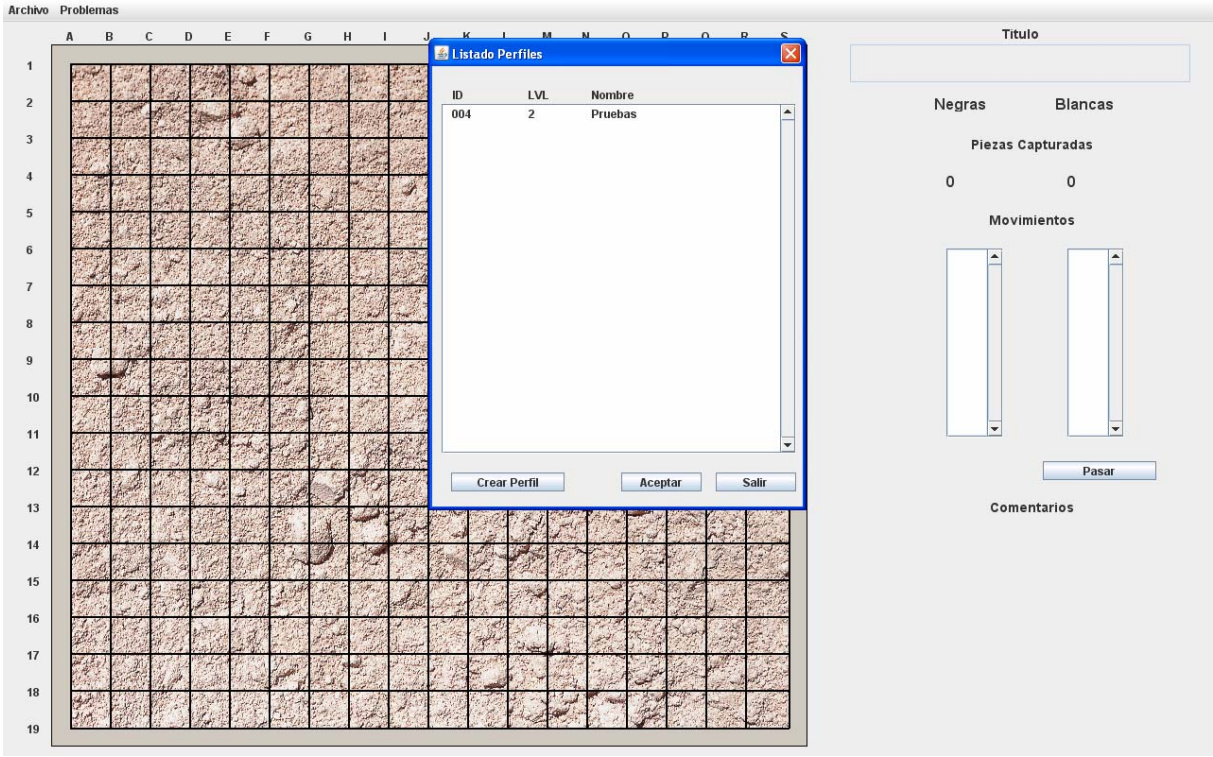
Ventana inicial:

Esta es la ventana inicial en la que podemos seleccionar las distintas funciones que nos ofrece:

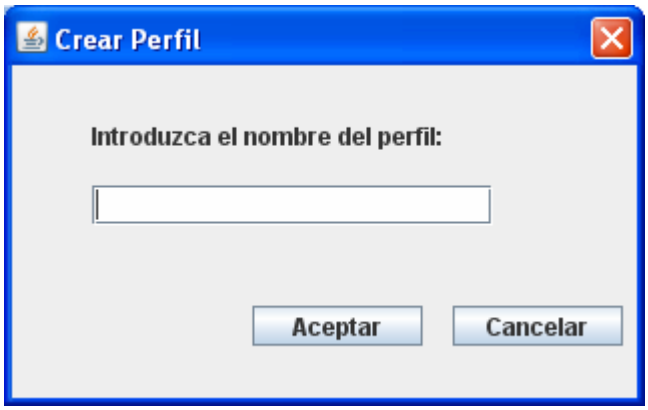
- Jugar
- Crear Problema
- Borrar Problema
- Borrar Perfil
- Exportar Problema
- Importar Problema
- Salir

Juego Go:

Cuando le damos a jugar nos aparece el siguiente diálogo, que nos permite elegir un perfil de usuario o crear un perfil de usuario:

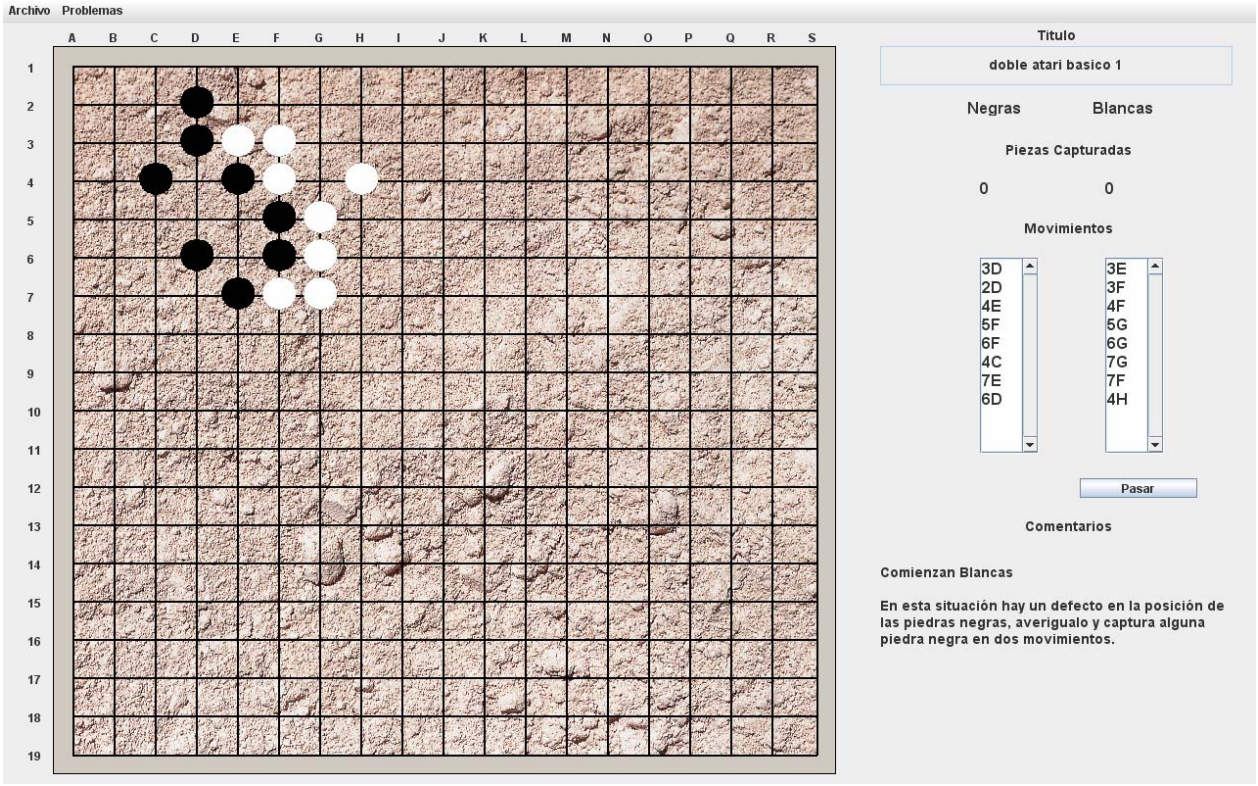


Como podemos observar el interfaz es bastante intuitivo. Si le damos a "Crear Perfil" se despliega la siguiente ventana:



Al introducir el nombre de perfil y darle al botón de "Aceptar", lo almacenamos en la base de datos.

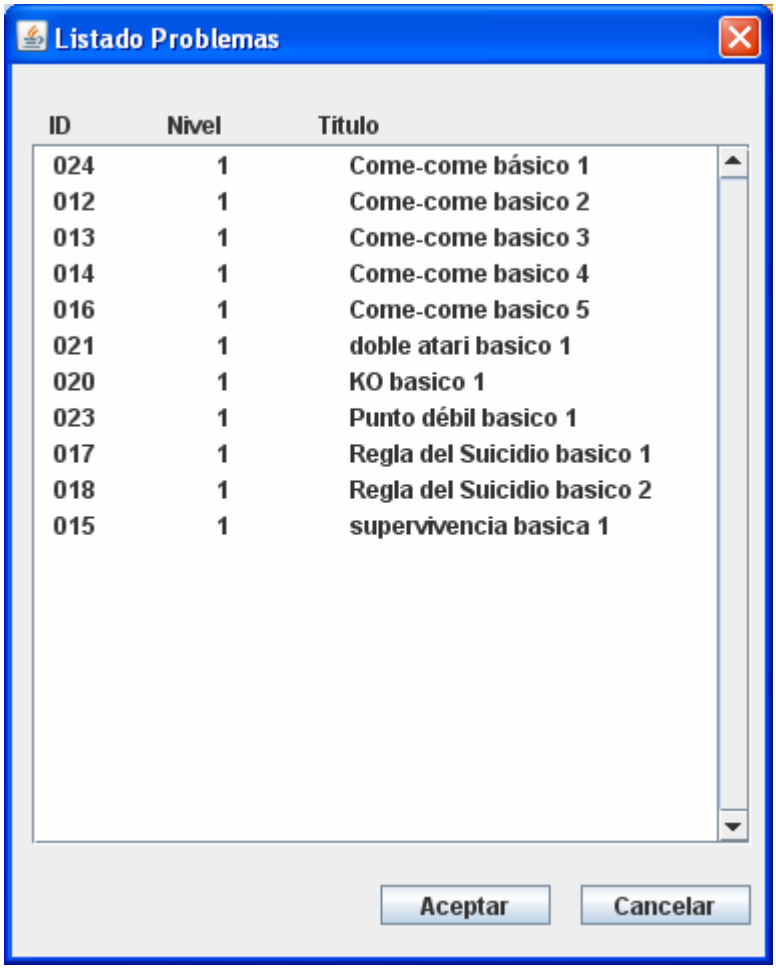
Este es el entorno de juego, en el que nada al mostrarse ya podremos empezar a jugar contra la máquina.


Archivo Problemas
A B C D E F G H I J K L M N O P Q R S
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
Titulo
doble atari basico 1
Negras
Blancas
Piezas Capturadas
0
0
Movimientos
3D 2D 4E 5F 6F 4C 7E 6D
3E 3F 4F 5G 6G 7G 7F 4H
Pasar
Comentarios
Comienzan Blancas
En esta situación hay un defecto en la posición de las piedras negras, averigualo y captura alguna piedra negra en dos movimientos.

Como podemos observar tenemos dos menús en la parte superior izquierda de la imagen, que nos permiten salir y seleccionar un problema de juego respectivamente.

En la parte derecha de la imagen tenemos el título del problema seleccionado, como se encuentra la situación de la partida y los comentarios correspondientes del problema.

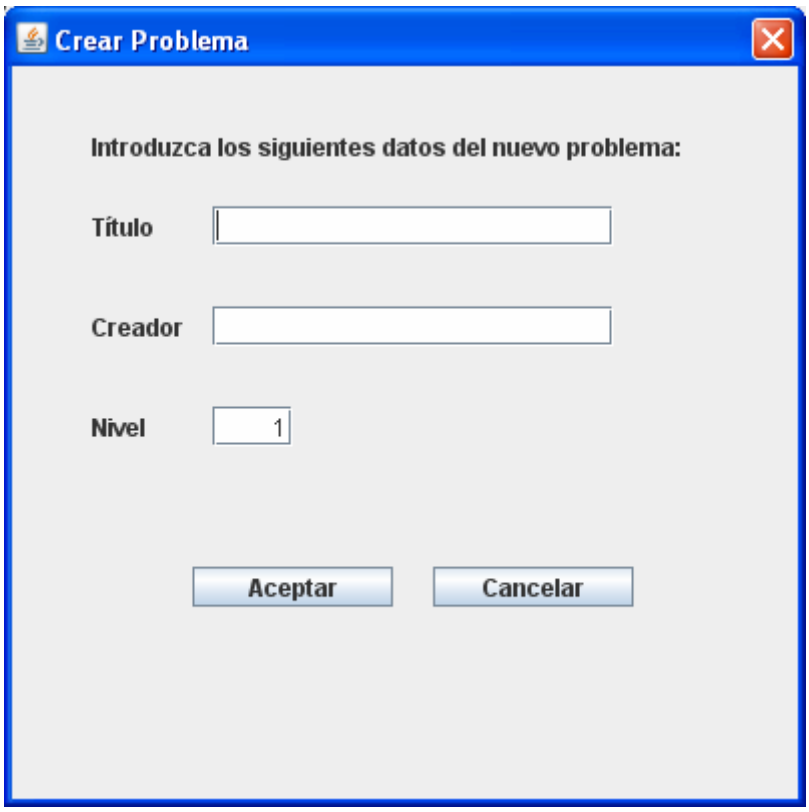
Para seleccionar un problema lo haremos a través del diálogo que mostramos a continuación


Listado Problemas
ID
Nivel
Titulo
024 1 Come-come básico 1
012 1 Come-come basico 2
013 1 Come-come basico 3
014 1 Come-come basico 4
016 1 Come-come basico 5
021 1 doble atari basico 1
020 1 KO basico 1
023 1 Punto débil basico 1
017 1 Regla del Suicidio basico 1
018 1 Regla del Suicidio basico 2
015 1 supervivencia basica 1
Aceptar
Cancelar

Una vez seleccionado el problema, le damos al botón aceptar, a continuación la aplicación carga en la aplicación el problema en cuestión.

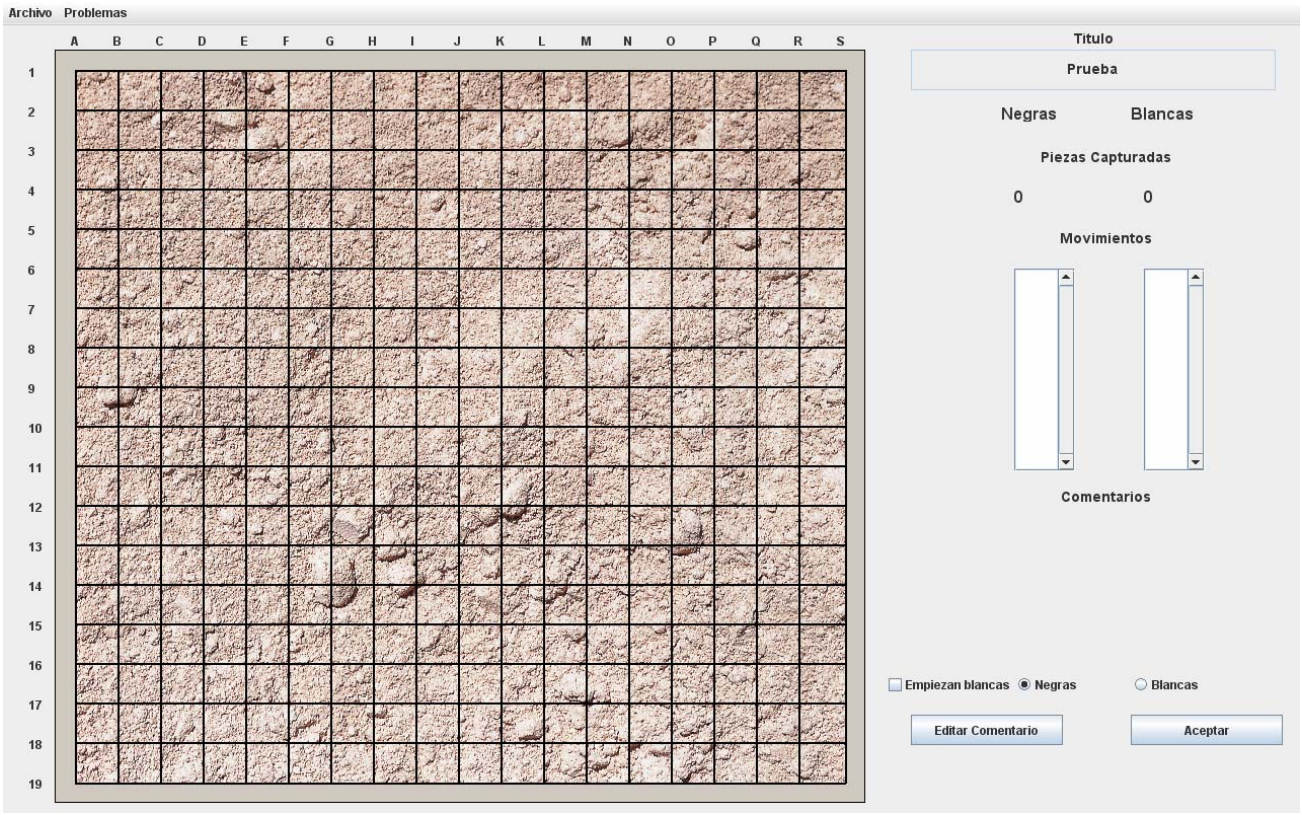
Una vez cargado el problema, mostramos el título y comentarios en el entorno de juego.

Esta ventana es la inicial que se presenta cuando un usuario quiere crear un problema, en ella el usuario debe introducir el título del problema que va a crear, el nombre del creador del problema, y asignarle un nivel al problema, por defecto el nivel asignado a un problema es 1.



Editor de problemas:

A continuación mostraremos la ventana principal en el editor de problemas:



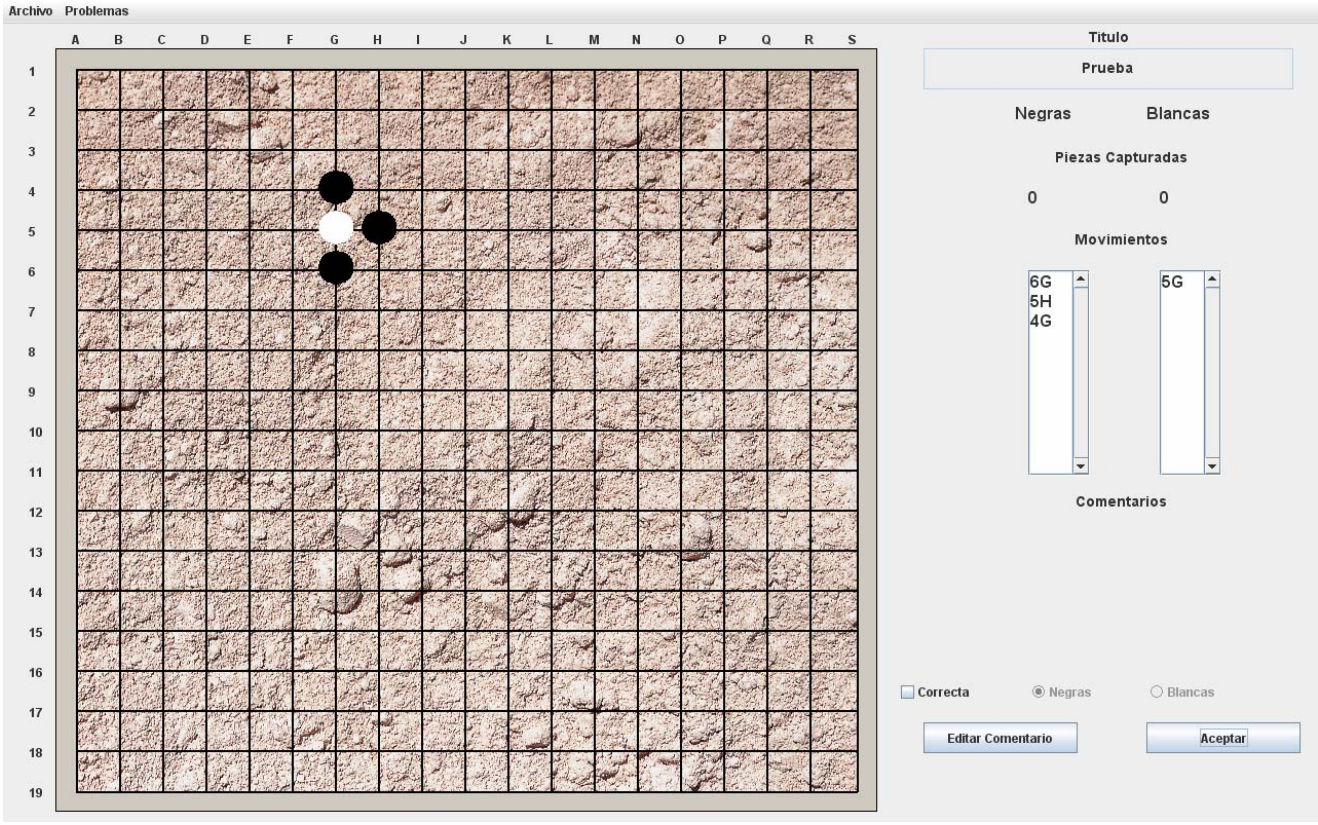
Esta ventana difiere en ventana de juegos en que en ésta tenemos los botones de aceptar y de editar comentar, el checkbox “Empiezan blancas” y los radiobuttons “Negras” y “Blancas”.

Comenzaremos introduciendo jugada inicial del problema en el tablero y añadiendo un comentario para que se muestre al comienzo del problema, a continuación se pulsa el botón “Aceptar” y guardamos en la aplicación el problema con la jugada inicial.

El checkbox “Empiezan blancas” si no se marcar, por defecto, empiezan las piezas negras, en caso contrario, empiezan las piezas blancas.

Los radiobuttons “Negras” y “Blancas” me permiten, seleccionando uno u otro colocar una pieza de uno u otro color en el tablero.

Ejemplo de jugada inicial:

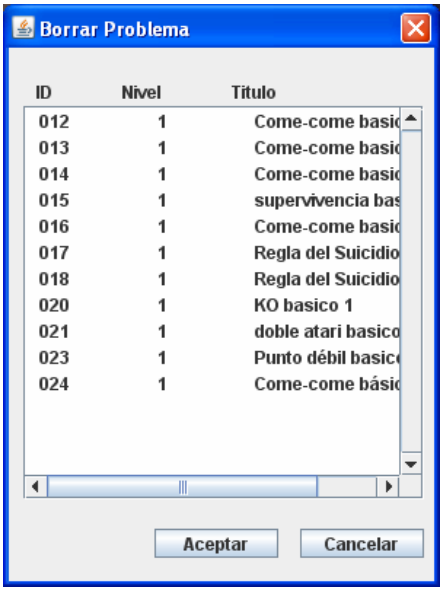


Como vemos después de introducir la jugada inicial el checkbox ha cambiado, ahora permite indicar si la jugada que vamos a introducir es correcta o no.

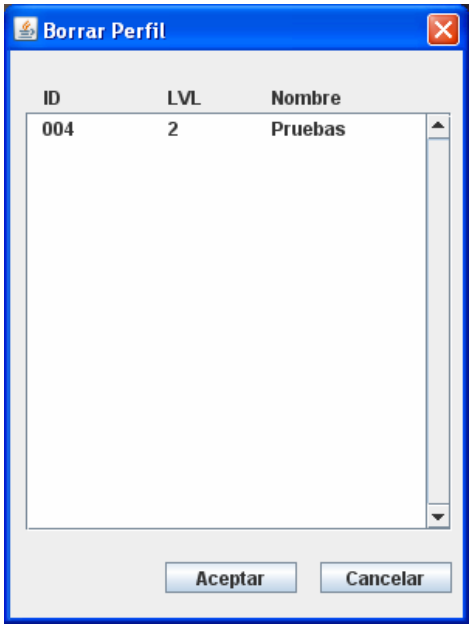
Una vez desarrollada una jugada, le damos a aceptar y volveremos a la situación inicial del problema lo que nos permitirá desarrollar más jugadas y añadir los respectivos comentarios.

En el menú superior tenemos la opción problema y dentro de ésta la opción “Guardar” que permite guardar el problema junto con todas las jugadas creadas en la base de datos.

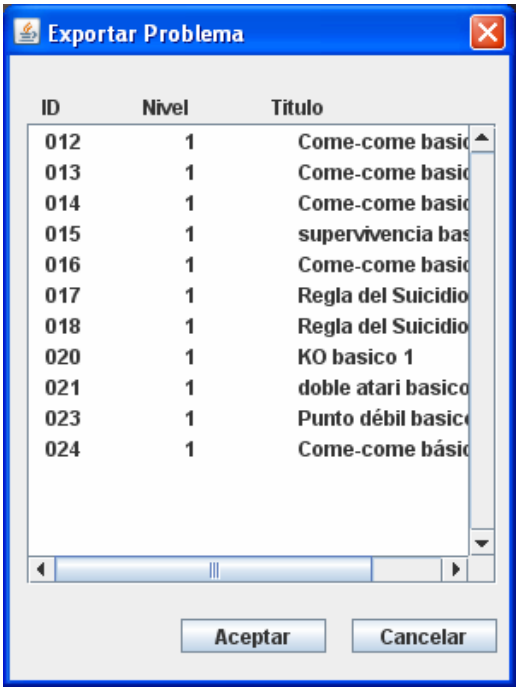
Diálogos para la gestión:



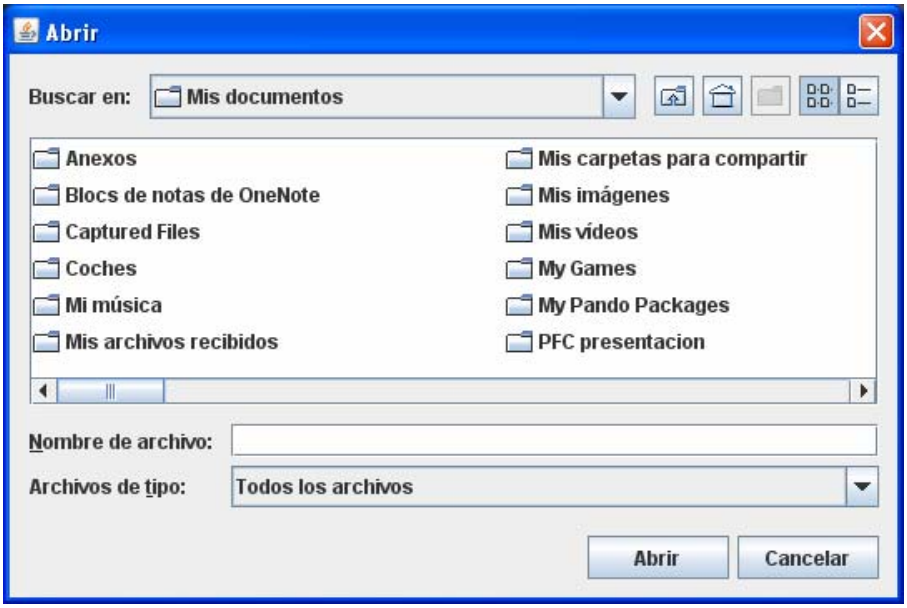
Este diálogo me permite, sólo con seleccionar un problema y pulsar “Aceptar”, eliminar un problema.



Este diálogo me permite, sólo con seleccionar un perfil y pulsar “Aceptar”, eliminar un perfil.



Este diálogo me permite, sólo con seleccionar un perfil y pulsar “Aceptar”, exportar un problema a un fichero externo que se almacena en el directorio que contiene la aplicación. La extensión que tiene el fichero es “.urb.



Este es el diálogo que nos permite importar problemas, con este diálogo seleccionamos el problema que queremos importar y automáticamente se almacena en la base de datos.